UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Controle Nebuloso e Algoritmos Genéticos: Uma aplicação ao robô móvel Khepera.

Dissertação submetida à Universidade Federal de Santa Catarina
como requisito parcial à obtenção do grau de

Mestre em Engenharia Elétrica

por

Amarilys Lima López

Florianópolis, 16 de abril de 1999

Controle Nebuloso e Algoritmos Genéticos: Uma aplicação ao robô móvel Khepera.

Amarilys Lima López

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Engenharia** na especialidade **Engenharia Elétrica**, área de concentração **Controle, Automação e Informática Industrial**, e aprovada em sua forma final pelo Programa de Pós-Graduação.

Florianópolis, 16 de abril de 1999.

Prof. Guilherme Bittencourt, Dr.
Orientador

Prof. Ildemar Cassana Decker, D. Sc.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Santa Catarina.

Banca Examinadora

Prof. Guilherme Bittencourt, Dr.
Orientador

Prof. Marcelo Ricardo Stemmer, Dr.

Prof. Sandra Aparecida Sandri, Dra.

à minha filha Anais,

ao meu esposo Alejandro,

aos meus pais Adelina e Misael...

Agradecimentos

Ao meu orientador, o professor Guilherme Bittencourt, pela dedicação, competência e bom caráter mostrados na orientação deste trabalho.

Aos membros da banca examinadora pelas contribuições ao aprimoramento da dissertação.

À CAPES, por conceder-me a bolsa de estudos.

Aos alunos de Iniciação Científica Valdemar Antonio Dallagnol Filho e Luciano Rottawa da Silva, pela ajuda no desenvolvimento do trabalho.

Ao colega Leandro dos Santos Coelho, por sua constante ajuda e amizade.

A Monica e o Augusto por sua ajuda tanto social como pessoal.

A todos os colegas do laboratório, principalmente: Karina, Marcos, Sandra, Jorge Ricardo, Sandro, Ronei, Carlos Rocha, Cesar, Fili, Antonio Eduardo Carrilho, Ivana e família, pela amizade e companheirismo.

A Fonseca e família, Taboada e família, Rene (obrigado pela filmada), Tesi e Pedro, e a todos os cubanos que estão perto de nós.

A Hugo e a Rita que contribuíram com as correções gramaticais do texto final.

Agradeço a meu primo Abiel Roche, por sua ajuda (cuidando de minha filha) no final da dissertação.

A minha família pelo carinho e apoio, mesmo que longe, sempre estiveram presentes: minhas irmãs Anitsa e Anoland, meus Pai e Mãe, minha sobrinha Maria Carla e meu sobrinho Mauricio.

As minhas amigas Niurkita, Esperancita e Tania, que sempre me escrevem e me têm presente como eu a elas.

Em especial a Alejandro García, pelo amor e carinho, e o incessante estímulo.

Resumo

Os Controladores Lógicos Nebulosos são cada vez mais utilizados com sucesso em diferentes aplicações industriais. De forma similar, os procedimentos de otimização baseados na teoria da evolução das espécies, onde estão incluídos nos Algoritmos Genéticos (AGs), têm merecido bastante interesse da comunidade científica nos últimos anos. O objetivo principal desta dissertação constitui o desenvolvimento de módulos de controle robustos baseados na Lógica Nebulosa e os Algoritmos Genéticos para realizar duas tarefas de navegação em robótica móvel: Evitar Obstáculos e Seguir Paredes. Estas técnicas de controle são implementadas no simulador do robô móvel Khepera. São elaboradas as funções de pertinência das entradas, saídas e as bases de regras de inferência para cada tarefa de navegação. São analisadas quatro metodologias de otimização usando um algoritmo genético, e implementados três métodos de seleção: Roleta, Estocástico e Torneio. É realizada uma comparação dos resultados obtidos com o controlador nebuloso e o controlador nebuloso otimizado para Seguir Paredes através de um algoritmo genético.

Abstract

Fuzzy Logic Controllers (FLCs) are being widely and successfully used in different areas. Besides, the application of robust methodologies based on probabilistic search and optimization procedures, like Genetics Algorithms (GAs), become a great trend in the research community. This work is concerned with the study and applications of FLCs and GAs to produce robust basic behavior modules for mobile robot navigation: Avoid and Follow Wall. Simulations of these metodologies are performed using the Khepera's Simulator. Fuzzy membership functions and fuzzy rules are developed for both navigations tasks. Four otimization metodologies and tree selection methods are analyzed using a genetic algoritm: Ruleta, Stochastic and Tournament. The performance of the fuzzy controller and of the otimized fuzzy logic controller to Follow Wall are analyzed.

Sumário

Capítulo 1

Introdução

Desde a década passada, a lógica Nebulosa tem merecido um grande interesse na comunidade científica. Uma das razões foi o sucesso na indústria japonesa. Já em 1990 o mercado dos controladores nebulosos tinha sido ampliado a um número crescente de aplicações industriais. Atualmente, baseada nos trabalhos pioneiros de Lofti A. Zadeh [Zad65], uma abundante literatura trata os aspectos teóricos e industriais da lógica nebulosa.

Os controladores nebulosos pertencem a uma classe de *sistemas baseados em conhecimento*. O principal objetivo é o de implementar o conhecimento humano na forma de regras lingüísticas *se - então*. A lógica nebulosa proporciona o formalismo matemático para alcançar este objetivo. Mas nem tudo está resolvido em relação à lógica nebulosa. Um dos problemas a resolver ainda é a falta de uma metodologia sistemática no desenvolvimento do controlador nebuloso e a dificuldade para analisar o seu desempenho [God97]. Assim, por exemplo, a não linearidade afeta a resposta do sistema quando o controlador nebuloso possui um grande número de parâmetros que devem ser ajustados. O ajuste dos parâmetros pode consumir muito tempo e requerer engenheiros com suficiente experiência em controle. Uma solução seria o uso de técnicas de estimação baseadas no *controle adaptativo* [AW89], onde os parâmetros do controlador são atualizados durante a operação da planta. Estes parâmetros também podem ser obtidos através de diferentes técnicas de aprendizado tais como: as redes neurais artificiais, os sistemas neuro-nebulosos, os algoritmos genéticos e evolutivos, a técnica "simulated annealing", entre outras.

Sem dúvida a otimização do controle nebuloso tem sido um dos principais tópicos de interesse da comunidade científica. O objetivo é desenvolver módulos de controle cada vez mais robustos. Em particular, uma das técnicas que tem recebido especial interesse nos últimos anos no campo da Inteligência Artificial são os algoritmos genéticos (AGs). Os AGs são procedimentos de otimização probabilísticos baseados na teoria da evolução das

espécies, os quais trabalham com cadeias finitas de bits que representam o conjunto de parâmetros de interesse do problema em questão e onde uma função de avaliação avalia cada uma destas cadeias.

Por outro lado, um robô móvel é uma planta não linear difícil de modelar. Portanto, constitui um bom candidato como plataforma experimental para a validação das técnicas de controle mencionadas. Além disto, as variáveis de estado de um robô móvel (a posição no espaço cartesiano e a orientação) são simples para se visualizar pois elas tem uma relação intuitiva com sua conduta. Assim, as regras lingüísticas do tipo *se - então* podem ser também definidas de maneira intuitiva. Porém, quando o número de sensores e atuadores é elevado, a complexidade do controle é maior, dificultando-se a construção da base de regras, especialmente se são desejados comportamentos complexos. Como tarefas complexas são construídas sobre a base de tarefas mais simples, poderia ser desejável testar as possibilidades dos controladores nebulosos e dos algoritmos genéticos iniciando-se em tarefas de baixo nível.

1.1 Objetivos

Este trabalho tem como objetivo o estudo de duas metodologias de controle: o Controle Fuzzy e os Algoritmos Genéticos. Será explorada a utilização destas técnicas no controle de Robôs Móveis Autônomos. Assim, é possível destacar os seguintes pontos de interesse:

- O desenvolvimento de um controlador nebuloso que permita modelar tarefas básicas de navegação em robótica móvel.
- A otimização do controlador nebuloso usando-se um algoritmo genético.
- A implementação destas técnicas utilizando simulações.

A plataforma para a implementação (simulação) dos controladores será o simulador do robô móvel Khepera [MAG95].

1.2 Estrutura da Dissertação

Esta dissertação está estruturada em sete capítulos. O Capítulo 2 faz uma breve introdução aos temas básicos deste trabalho: o tratamento de incertezas e a computação evolutiva.

O Capítulo 3 descreve os principais tópicos relacionados à lógica nebulosa. São apresentados os conceitos básicos da teoria de conjuntos nebulosos e os operadores. Especial

interesse é dado aos aspectos do desenvolvimento de controladores nebulosos e as etapas que o formam. São tratados vários exemplos para facilitar a compreensão destes tópicos.

No Capítulo 4 são tratados os Algoritmos Genéticos (AGs). São apresentadas as definições mais importantes, os operadores genéticos e os passos do algoritmo.

O Capítulo 5 introduz brevemente os aspectos mais importantes do controle de Robôs Móveis Autônomos. Em particular são tratados dois comportamentos básicos: evitar obstáculos (Avoid) e seguir paredes (Follow Wall).

No Capítulo 6 são mostrados os resultados das implementações dos controladores estudados através de simulações. Foi utilizado o robô móvel Khepera como plataforma de experimentação.

O conteúdo desta dissertação é completado com o Capítulo 7, onde são analisadas as metas alcançadas e são identificadas as perspectivas para trabalhos futuros.

Capítulo 2

Breve introdução à Incerteza e à Computação Evolutiva

"I believe that nothing is unconditionally true, and hence I am opposed to every statement of positive truth and every man who makes it."

H. L. Mencken [Mat97].

2.1 Introdução

Neste capítulo é apresentada uma revisão de artigos sobre dois temas da IA considerados importantes para o objetivo de introduzir os principais conceitos, teorias e técnicas associadas aos temas que serão abordados neste trabalho e propôr leituras complementares através da bibliografia. Este texto está baseado no capítulo 6 do livro "Inteligência Artificial. Ferramentas e Teorias"[Bitt98a].

Os tópicos escolhidos e as razões para sua escolha são : (i) *representação de incerteza*, pelo papel estratégico reservado ao tratamento da incerteza, pois a IA trata exatamente daqueles problemas que devido à sua complexidade e/ou ao seu tamanho, tornam-se intrinsecamente incertos; e (ii) *computação evolutiva*, pela novidade, engenhosidade e apelo intuitivo do tema, que ao propor o desenvolvimento de computações artificiais inspiradas nas "computações" biológicas realizadas pelos seres vivos para viver e se reproduzir, abre um enorme campo de pesquisa abrangendo desde aplicações práticas, como a solução de problemas complexos, até a exploração de um domínio totalmente novo, a *vida artificial* [Bitt98a].

2.2 Representação de incerteza

A imperfeição da informação é geralmente conhecida na literatura de sistemas baseados em conhecimento como *incerteza* [San97]. No entanto, este termo é muito restritivo; o que se convenciona chamar tratamento de incerteza pode, na verdade, estar endereçando outras imperfeições da informação, como imprecisão, conflito, ignorância parcial, etc.

Suponhamos, por exemplo, que queiramos descobrir a que horas começa um determinado filme. Algumas das respostas que podemos obter são [San97]:

- Informação perfeita: O filme começa às 8h15min.
- Informação imprecisa: O filme começa entre 8h e 9h.
- Informação incerta: Eu acho que o filme começa às 8h (mas não tenho certeza).
- Informação vaga: O filme começa lá pelas 8h.
- Informação probabilista: É provável que o filme comece às 8h.
- Informação possibilista: É possível que o filme comece às 8h.
- Informação inconsistente: Maria disse que o filme começa às 8h, mas João disse que ele começa às 10h.
- Informação incompleta: Eu não sei a que horas começa o filme, mas usualmente os filmes neste cinema começam às 8h.
- Ignorância total: Eu não faço a menor idéia do horário do filme.

As informações que podemos obter podem, portanto, variar de perfeitas, quando descobrimos exatamente o que queremos saber, a completamente imperfeitas, seja pela total ausência de informações ou por informações completamente conflitantes. O mais interessante aqui é que, mesmo lidando diariamente com o tipo de informações acima, conseguimos tomar decisões razoáveis. Para tanto, nós, de alguma forma, encontramos um modelo adequado para representar a informação que obtivemos e a tratamos segundo o modelo escolhido. O mesmo deve (ou pelo menos deveria) ocorrer com sistemas baseados em conhecimentos, em face de informações imperfeitas [Bitt98a].

Por exemplo, para cada um dos tipos de informação descritos acima, existe um modelo formal conhecido de tratamento. Além disso, encontramos na literatura implementações formais ou "ad hoc" para cada um desses modelos. A informação de conotação probabilista pode ser tratada pela teoria de probabilidades ou pela teoria da evidência (também

conhecida como Dempster-Shafer) [Sha76]; enquanto que a informação imprecisa e/ou vaga pode ser tratada pela teoria dos conjuntos nebulosos ([Zad65], [DP80]), pela teoria dos *conjuntos de aproximação* (do inglês, "rough sets") [Paw92] ou pela manipulação de classes de referência [KJ83]; e a informação possibilista pode ser tratada pela teoria de possibilidades ([Zad78], [DP88]). A informação incerta pode ser tratada tanto pelas teorias de probabilidades, possibilidades ou evidência, ou por modelos "ad hoc". As informações inconsistentes e aquelas que chamamos, um pouco impropriamente, incompletas, podem ser tratadas por lógicas não clássicas; como a paraconsistente e a de quatro valores no primeiro caso [Bel77], e as lógicas não monotônicas no segundo caso, como a lógica de "default" [Rei80] e a circunscrição [McC80].

Além dos modelos mencionados, existem outros reconhecidamente importantes, como os modelos de propagação local de informação imperfeita ([Pea87], [San93]) e os modelos para agregação de opiniões de especialistas ([SDP95]).

Como o interesse fundamental deste trabalho centra-se no tratamento da informação imprecisa e/ou vaga, no capitulo 3 serão tratados os aspectos mais importantes relacionados à lógica nebulosa, tais como conjuntos nebulosos, operadores, regras de inferências, controle nebuloso e suas aplicações.

2.3 Computação evolutiva

A *computação evolutiva (CE)* é um novo paradigma da ciência da computação, que diferentemente do processado de dados convencionais, não exige para resolver um problema, o conhecimento prévio de uma maneira de encontrar uma solução. A CE é baseada em mecanismos evolutivos encontrados na natureza, tais como a auto-organização e o comportamento adaptativo [FTW83], [GH88]. Estes mecanismos foram descobertos e formalizados por Darwin em sua *teoria da evolução natural*, segundo a qual, a vida na terra é o resultado de um processo de seleção pelo meio ambiente, dos mais aptos e adaptados, e por isto mesmo com mais chances de reproduzir-se. A diversidade da vida, associada ao fato de que todos os seres vivos compartilham uma bagagem genética comum, pelo menos em termos de seus componentes básicos, é um exemplo eloqüente das possibilidades do mecanismo de evolução natural.

Segundo Heitkoetter [HB94], a CE pertence ao ramo da *computação natural* que inclui os tópicos de vida artificial, geometria fractal, sistemas complexos e inteligência computacional. Este último inclui redes neuronais artificiais, sistemas nebulosos (do inglês "fuzzy systems") e a CE.

Historicamente as primeiras iniciativas na área de CE foram de biólogos e geneticistas

interessados em simular os processos vitais em computador, o que recebeu na época o nome de "processos genéticos". Alguns desses cientistas, citados em [Gol89], são: Barricelli, 1957, 1962; Fraser, 1960, 1962; Martin e Cockerham, 1960. Em sua tese de doutorado, o biólogo Rosenberg, em 1967, simulou uma população de seres unicelulares, estrutura genética clássica (um gene, uma enzima) com estrutura diplóide, com cromossomos de 20 genes e 16 alelos permitidos em cada um [Gol89].

Já na década de 60, Holland e outros começaram a estudar os chamados *sistemas adaptativos*, que foram modelados como sistemas de *aprendizagem de máquina*. Tais modelos, conhecidos como *algoritmos genéticos*, implementavam populações de indivíduos contendo um genótipo, formado por cromossomos (que neste modelo eram representados por cadeias de "bits") aos quais se aplicavam operadores de seleção, recombinação e mutação. Ainda que Holland tenha proposto um quarto operador (a inversão), este não chegou a ser largamente utilizado [Dav91].

Uma das primeiras aplicações propostas para os algoritmos genéticos, seguindo o enfoque de Holland, foram os sistemas classificadores, que são sistemas de produção e, na verdade, usam os algoritmos genéticos em uma parte do algoritmo global. Apesar do interesse que levantaram na época, os sistemas classificadores permanecem como um campo de estudo em grande parte ainda inexplorado.

Outro ramo descendente dos algoritmos genéticos é o da *programação genética (PG)* [Gol89]. Aqui, os indivíduos da população não são seqüências de "bits", mas sim programas de computador armazenados na forma de árvores sintáticas. Tais programas é que são os candidatos à solução do problema proposto. A programação genética não usa o operador mutação e a recombinação se dá pela troca de subárvores entre dois indivíduos candidatos à solução.

Outro ramo da CE é a *programação evolutiva (PE)* [Gol89], que visa prever o comportamento de máquinas de estado finito. Apenas dois operadores são usados: a seleção e a mutação. As idéias datam de 1966, e não foram muito consideradas na comunidade de CE por rejeitar o papel fundamental da recombinação. Finalmente, um último ramo é a *estratégia evolutiva* [Gol89], proposta nos anos 60, na Alemanha. A ênfase aqui é na auto-adaptação. O papel da recombinação é aceito, mas como operador secundário.

Embora tenham origens bastante diversas, todas essas abordagens têm em comum o modelo conceitual inicial – a evolução natural –, além dos operadores e, mais importante, o mesmo objetivo final: a solução de problemas complexos [BS93].

2.3.1 Idéias básicas da CE

A CE está baseada em algumas idéias básicas, que quando implementadas, permitem simular em um computador o processo de passagem de gerações da evolução natural. As idéias que permitem esta simulação são as seguintes [Bitt98a]:

- A criação de uma população de soluções, possivelmente obtida na sua primeira geração de modo aleatório, e na qual os indivíduos tenham registrado de modo intrínseco os parâmetros que descrevem uma possível solução ao problema posto.
- A criação de uma entidade – chamada *função de avaliação* – capaz de julgar a aptidão de cada um dos indivíduos. Essa entidade não precisa deter conhecimento sobre como encontrar uma solução para o problema, mas apenas atribuir uma "nota" ao desempenho de cada um dos indivíduos da população.
- E, finalmente, a criação de uma série de operadores que serão aplicados à população de uma dada geração para obter os indivíduos da próxima geração. Estes operadores são baseados nos fenômenos que ocorrem na evolução natural. Os principais operadores citados na literatura são: (i) *seleção*: permite escolher um indivíduo ou um par deles para gerar descendência. Note-se que este operador simula a reprodução assexuada (no primeiro caso) e a sexuada (no segundo) que ocorrem na natureza. Obviamente, a prioridade da escolha recai sobre os indivíduos mais bem avaliados pela função de avaliação; (ii) *recombinação*: operador que simula a troca de material genético entre os ancestrais que, por sua vez, determina a carga genética dos descendentes; (iii) *mutação*: operador que realiza mudanças aleatórias no material genético.

O conceito chave na CE é o de *adaptação*, que unifica a abordagem quanto ao método de solução: uma população inicial de soluções evolui, ao longo das gerações que são simuladas no processo, em direção a soluções mais adaptadas, isto é, com maior valor da função de avaliação, por meio de operadores de seleção, mutação e recombinação [Bitt98a]. O conjunto de soluções iniciais pode ser aleatório ou pode ser obtido a partir de técnicas convencionais para resolver instâncias mais simples do problema que está sendo tratado. Por um lado, usando-se soluções inicialmente aleatórias, pode-se usar sempre o mesmo algoritmo e os mesmos operadores. Por outro lado, adaptando o conceito de CE a um problema específico e empregando soluções iniciais obtidas por métodos convencionais, é necessário adaptar os operadores usuais da CE para o problema específico. Em compensação, na pior das hipóteses, a solução encontrada é igual à melhor solução obtida anteriormente pelas técnicas convencionais.

Para definir a função de avaliação é necessário encontrar uma maneira de codificar as soluções para o problema que se quer resolver. O resultado dessa codificação corresponde aos cromossomos na evolução natural e é chamado de *genótipo*. A partir desses cromossomos, a função de avaliação deve ser capaz de determinar a qualidade de uma solução.

As novas soluções podem ser geradas a partir de uma única solução (assexuada, na natureza) ou a partir de duas soluções (na natureza sexuada). Estabelecido um conjunto de novas soluções (os descendentes), estas sofrem a ação dos operadores evolutivos, mediante os quais, os descendentes passarão a ser diferentes dos ascendentes. Os melhores, de acordo com a função de avaliação, terão uma descendência maior do que a dos pouco aptos. Na reprodução sexuada, a troca de material genético – chamada de *recombinação* – leva um par de ascendentes a dar origem a um par de descendentes onde cada descendente herda partes aleatoriamente escolhidas de cada ascendente. A mutação leva à mudança, também aleatória, de uma parte da solução. No caso mais simples de cromossomos codificados em binário, a mutação é a simples inversão de um "bit". Tanto a recombinação quanto a mutação tendem a ocorrer segundo uma dada probabilidade (que são parâmetros da técnica).

Tal processo, repetido, simula a passagem das gerações. Como o processo está sendo simulado em um computador digital, o fator tempo pode ser comprimido sem perda de qualidade.

2.3.2 Formalização

Uma possível formalização algorítmica para a CE foi proposta por [BS93]. Seja I o conjunto de indivíduos e $\Re$ o conjunto dos números reais. $\alpha(t) \in I$ denota um indivíduo na geração t e $\vec{x} \in \Re^n$ indica as coordenadas de um ponto no espaço de soluções. $\mu \geq 1$ denota o tamanho da população de ancestrais, e $\lambda \geq 1$ é o tamanho da população de descendentes. Definem-se as seguintes funções:

Função objetivo a ser otimizada: $f : \Re^n \rightarrow \Re$.

Função de avaliação: $\Phi : I \rightarrow \Re$.

A população na geração t, denotada por $P(t) = \{\alpha_1(t), \ldots, \alpha_\mu(t)\}$ consiste de indivíduos $\alpha_i(t) \in I$, onde cada indivíduo codifica com um grau de qualidade dado por $\Phi(\alpha_i(t))$, a função objetivo f. Os operadores são definidos da seguinte maneira:

Seleção: $s_{\Theta_s} : (I^\lambda \cup I^{\lambda+\mu}) \rightarrow I^\mu$.

Recombinação: $r_{\Theta_r} : I^\mu \rightarrow I^\lambda$.

Mutação: $m_{\Theta_m} : I^\lambda \rightarrow I^\lambda$.

Os operadores são controlados, respectivamente, pelos conjuntos de parâmetros Θ_r, Θ_m e Θ_s. Durante a etapa de avaliação, a função de avaliação Φ é calculada para todos os elementos da população. O critério de fim é dado por:

$$\iota : I^\mu \rightarrow \{V, F\}$$

onde V e F são valores verdade. Finalmente, Q é o conjunto de indivíduos que são tomados adicionalmente durante a etapa de seleção. Se Q é vazio, os indivíduos de uma geração não são usados como candidatos ascendentes na próxima geração. Se $Q = P(t)$, todos os indivíduos de uma geração não são utilizados na próxima. De acordo com essa notação, pode-se descrever o algoritmo básico da CE da seguinte maneira [Bitt98a]:

1. $t \leftarrow 0$
2. População inicial: $P(0) \leftarrow \{\alpha_1(0), \ldots, \alpha_\mu(0)\} \in I^\mu$
3. Avaliação: $\{\Phi(\alpha_1(0)), \ldots, \Phi(\alpha_\mu(0))\}$
4. Enquanto $\iota(P(t)) \neq V$ faça:
   - Recombinação: $P'(t) \leftarrow r_{\Theta_r}(P(t))$
   - Mutação: $P'(t) \leftarrow m_{\Theta_m}(P'(t))$
   - Avaliação: t
   - Seleção: $P(t+1) \leftarrow s_{\Theta_s}(P'(t) \cup Q)$
   - $t \leftarrow t+1$

Dentro da CE existem diversas correntes ou abordagens. Entre elas, a Estratégia Evolutiva, a Programação Evolutiva e os Algoritmos Genéticos são as mais conhecidas (ver [Bitt98a]). Todas compartilham a simulação do processo de seleção natural, mas se diferenciam pela origem e por suas estratégias de solução.

No Capitulo 4 serão tratados os aspectos mais importantes relacionados aos Algoritmos Genéticos [Bitt98a].

2.4 Conclusão

Neste capítulo foram abordados diferentes tópicos de interesse da Inteligência Artificial que serão úteis para o desenvolvimento deste trabalho. Em particular, o tratamento de incertezas, a lógica nebulosa e os algoritmos genéticos serão tratados com maior profundidade nos seguintes capítulos. Uma grande área de pesquisa e de aplicações continua aberta nestes campos (ver [Sch91], [Gol94], [Len95], [Red95]), entre outros.

Capítulo 3

Lógica Nebulosa

"So far as the laws of mathematics refer to reality, they are not certain. And so far as they are certain, they do not refer to reality."

Albert Einstein [Mat97].

3.1 Introdução

Observando que muitos conceitos no mundo real não podem ser bem representados usando limites claramente definidos, Lofti A. Zadeh desenvolveu a teoria dos conjuntos nebulosos [Zad65]. Esta teoria generaliza a teoria clássica dos conjuntos para permitir que os objetos possuam graus de pertinência à determinados conjuntos, possibilitando assim, a representação de conceitos vagos e imprecisos, porém, mantendo a precisão matemática no tratamento.

Com a lógica nebulosa é possível desenvolver um sistema especialista capaz de imitar o comportamento de um ser humano experimentado, sem a necessidade do estabelecimento preciso do modelo matemático do sistema a ser controlado [YZ91]. É portanto, indicada para processos complexos onde uma representação matemática torna-se complexa demais, tais como processos que envolvem reações químicas, fabricação de cimento, controle de vôo de aeronaves, instrumentação médica, sistemas de decisão, etc.

A lógica Nebulosa tem diferentes significados. Em um sentido restrito pode ser vista como um sistema lógico, como extensão da lógica *multivalores* (LM). Porém, em um sentido mais amplo é sinônimo da *teoria de conjuntos nebulosos* (TCN), relacionada a classes de objetos sem transições bruscas, nos quais uma função de pertinência representa uma variação gradual. Nesta perspectiva, a lógica nebulosa (LM), num sentido mais restrito, é o cérebro da lógica nebulosa num sentido mais amplo (TCN) [JVB94].

A lógica está relacionada à importância relativa da precisão [BNW96]) , conforme afirmação de Lofti Zadeh [Zad65]: "Quando a complexidade aumentar, afirmações precisas perdem significado e afirmações significativas perdem precisão"[1].

Neste capítulo serão abordados os aspectos mais importantes relacionados à lógica nebulosa, tais como: os conjuntos nebulosos, as operações entre conjuntos, as regras de inferência, as etapas do controle seguindo esta metodologia, e por último serão mencionadas algumas aplicações. São apresentados exemplos para facilitar a compreensão dos tópicos que serão abordados.

3.2 Por que usamos a Lógica Nebulosa?

A lógica Nebulosa é uma maneira conveniente de se realizar um mapeamento entre um espaço de entrada e um espaço de saída. Este mapeamento pode ser observado nos seguintes exemplos:

- Em função da qualidade do serviço num dado restaurante, assim será o valor da gorjeta;
- Em função de quanto quente se deseje a água, assim será o ajuste da temperatura;
- Em função do afastamento do alvo da foto, assim será o ajuste das lentes da câmara.

Em geral, o mapeamento pode ser representado por sistemas nebulosos, sistemas lineares, sistemas especialistas, redes neurais, etc. Em muitas aplicações práticas o sistema nebuloso é a melhor opção. A seguir são apresentadas algumas razões que justificam o uso da lógica nebulosa [Mat97]:

- Os conceitos matemáticos associados à lógica nebulosa são simples;
- É flexível;
- É tolerante a dados imprecisos;
- Permite modelar funções não lineares de complexidade arbitrária;
- Pode ser construída usando a experiência de especialistas;
- Muitas vezes é a opção de controle mais simples;

[1] "As complexity rises, precise statements lose meaning and meaningful statements lose precision"

- Está baseada na linguagem natural, por isso suas bases são as mesmas da comunicação humana.

No entanto, deve ser observado que nem sempre a lógica nebulosa é a melhor opção. Assim, deve ser tentada outra solução quando não existe um mapeamento *conveniente* entre os espaços de entrada e saída, ou quando o modelo matemático do sistema a ser controlado seja bem conhecido, ou quando existam outras soluções mais simples para resolver o mesmo problema [Mat97].

3.3 Conjuntos Nebulosos

Em processos onde existe um controle digital, um determinado elemento somente pode pertencer ou não a um determinado conjunto, não existindo valores entre verdadeiro e falso. Assim, a *lógica de Boole* somente retorna valores 0 ou 1. Já a lógica nebulosa substitui o *verdadeiro* e o *falso* por um conjunto contínuo de valores de pertinência compreendidos entre 0 e 1, refletindo o grau em que o elemento pertence ao conjunto.

A seguir serão apresentados vários exemplos sobre conjuntos nebulosos, mas primeiramente será definido um conceito importante da lógica nebulosa, o de *variável lingüística.*

Variável Lingüística: É definida pela quíntupla (X, T(X), U, G, M). Onde X é o nome da variável; T(X) é o conjunto dos nomes dos valores lingüísticos de X; U é o universo de discurso, G é a gramática para gerar os nomes; e M é o conjunto de regras semânticas que permitem associar a cada elemento X um significado.

Exemplo 2.1 [BNW96]:

Seja X um conjunto dos números reais entre 0 e 10, que será chamado de universo de discurso. Seja A um subconjunto de X constituído de todos os números reais entre 5 e 8:

A = [5,8]

A função característica de A associa um número (0 ou 1) a cada elemento em X, dependendo se o dito elemento pertence, ou não, ao subconjunto A. Isto pode ser observado na figura 3.1.

Pode-se interprer que os elementos com valor 1 pertencem ao conjunto A; porém, aqueles com valor 0, não pertencem ao conjunto A.

Mesmo que este conceito seja suficiente para muitas aplicações, ele evidencia perda de flexibilidade.

Exemplo 2.2 [BNW96]:

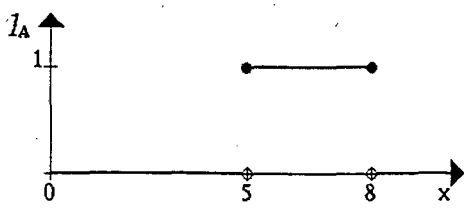


Figura 3.1: Função característica de A.

Seja B o conjunto de pessoas jovens (variável lingüística).

O limite inferior deste conjunto se inicia em 0, com o nascimento. Mais o limite superior não é simples de se estabelecer. Assim, como primeira tentativa, este valor será definido em 20 anos.

B = [0,20]

Então, como explicar que uma pessoa com 20 anos seja considerada como jovem e no dia seguinte não o seja mais? Para resolver este problema será relaxada a brusca separação entre jovem e não jovem. Assim, a decisão lógica SIM ele/ela pertence ao conjunto de pessoas jovens, ou NÃO, ele/ela não pertence à dito conjunto, será substituída por uma frase mais flexível como por exemplo: ele/ela pertencem, ligeiramente, ao subconjunto de pessoas jovens.

Na figura 3.2 mostra-se a função de pertinência de B. É possível observar a variação gradual da variável lingüística jovem.

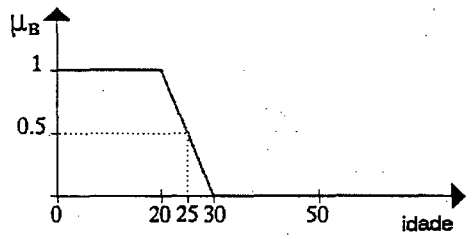


Figura 3.2: Função característica de B.

Neste exemplo a interpretação do universo de discurso é bem mais complexa. Novamente o valor 1 dado a um elemento significa que ele pertence ao conjunto B. Por outro lado, o valor 0 significa que ele não pertence ao conjunto B. Os outros valores da variável lingüística idade representam a pertinência gradual ao conjunto B.

Uma pessoa com 25 anos, é ainda jovem, com um grau de 50 %.

Exemplo 2.3 [San97]:

Suponhamos que queiramos modelar o conceito "alto" (variável lingüística). Usualmente, podemos dizer que uma pessoa que mede mais de $1,75m$ é alta, e que ela não é alta se tiver menos de $1,60m$. Já uma pessoa que mede entre $1,60m$ e $1,75m$ será considerada

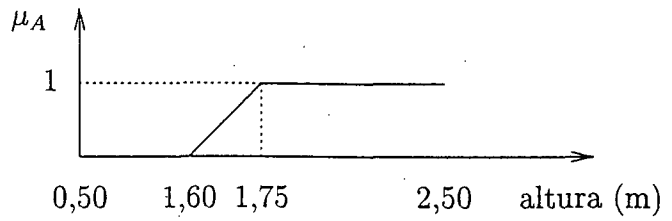


Figura 3.3: Conjunto nebuloso "alto"

tanto mais alta quanto mais sua altura esteja próxima de $1,75m$. Então podemos modelar o conceito "alto" pelo conjunto nebuloso A, definido no intervalo de $0,5m$ a $2,5m$, dado por:

$$\mu_A(x) = \begin{cases} 1 & x > 1,75m \\ 0 & x < 1,60m \\ \frac{x-1,6}{0,15} & 1,60m \leq x \leq 1,75m \end{cases}$$

A figura 3.3 ilustra o conjunto nebuloso "alto".

Assim, a cada pessoa que pertence ao universo de discurso será associado um grau de pertinência ao subconjunto nebuloso "alto". Isto foi possível definindo uma função de pertinência baseada na altura das pessoas.

3.4 Operações com conjuntos nebulosos

Similarmente às operações da lógica clássica, na lógica nebulosa são válidas as operações intersecção, união e negação. Estas operações permitem construir qualquer tipo de função lógica [BNW96].

Intersecção de Conjuntos: Sejam dois conjuntos A e B, então o conjunto resultante da intersecção deles, contém exatamente aqueles elementos que pertencem ao mesmo tempo aos conjuntos A e B.

União de Conjuntos: Sejam dois conjuntos A e B, então, o conjunto resultante da união deles contém todos os elementos de A, ou B, ou de ambos.

Negação de Conjuntos: Seja um conjunto A que pertence ao universo de discurso S, então, o conjunto resultante da negação de A está formado por todos os elementos que pertencem ao universo de discurso S, mas que não pertencem ao conjunto A.

Na teoria dos conjuntos nebulosos, a negação $n : [0,1] \rightarrow [0,1]$ é implementada por uma família de operadores, sendo que o mais comumente utilizado é dado por $n(x) = 1-x$.

Tabela 3.1: Principais T-normas e T-conormas duais

T-norma	T-conorma	Nome
$min(x, y)$	$max(x, y)$	Zadeh
$x \cdot y$	$x + y - xy$	Probabilista
$max(x + y - 1, 0)$	$min(x + y, 1)$	Lukasiewicz
$\frac{xy}{\gamma+(1-\gamma)(x+y-xy)}$	$\frac{x+y-xy-(1-\gamma)xy}{1-(1-\gamma)xy}$	Hamacher ($\gamma > 0$)
x, se $y = 1$ y, se $x = 1$ 0 se não	x, se $y = 0$ y, se $x = 0$ 1 se não	Weber

Ou seja, se $\overline{A}$ representa a negação de A no universo Y, então, utilizando a negação n temos $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$.

Nesta teoria, a intersecção é implementada por uma família de operações – chamadas *T-normas* – e a união é implementada por outra família de operações – chamadas *T-conormas* [DP88]. O conjunto das T-normas e T-conormas formam as *normas triangulares*. Cada norma triangular $\nu : [0, 1] \times [0, 1] \to [0, 1]$ verifica, para todos os x, y, z em $[0, 1]$, as seguintes propriedades [Bitt98a]:

(i) Comutatividade: $\nu(x, y) = \nu(y, x)$

(ii) Associatividade: $\nu(x, \nu(y, z)) = \nu(\nu(x, y), z)$

(iii) Monotonicidade: se $x \leq z$ e $y \leq t$, então $\nu(x, y) \leq \nu(z, t)$

Além disso, cada T-norma $\top$ verifica a propriedade:

(iv) $\top(x, 1) = x$ (elemento neutro 1),

e cada T-conorma $\perp$ verifica a propriedade:

(v) $\perp(x, 0) = x$ (elemento neutro 0).

Uma T-norma $\top$ e uma T-conorma $\perp$ são duais em relação a uma operação de negação n se elas satisfazem as relações de De Morgan, isto é, se $n(\top(A, B)) = \perp(n(A), n(B))$ e $n(\perp(A, B)) = \top(n(A), n(B))$. As T-normas e T-conormas duais mais utilizadas, em relação à operação de negação $1 - x$ são apresentadas no quadro 3.1.

É possível observar que estes operadores coincidem com as operações de união e intersecção da lógica clássica se, e somente se, são considerados os graus 0 e 1 das funções de pertinência.

Tabela 3.2: Principais operadores de implicação

Implicação	Nome
$max(1-x, y)$	Kleene
$min(1-x+y, 1)$	Lukasiewicz
1, se $x \leq y$ y, se não	Gödel
$min(x, y)$	Mandani
$x \cdot y$	Larsen

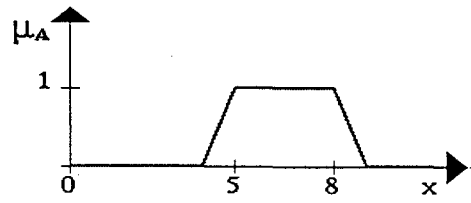


Figura 3.4: Intervalo nebuloso entre 5 e 8.

Outro operador usado na lógica nebulosa é a implicação. Os *operadores de implicação* $I : [0,1] \times [0,1] \rightarrow [0,1]$ são usados para modelar regras de inferência do tipo "Se ⟨premissa⟩ então ⟨conclusão⟩". Seja α o grau de compatibilidade entre as condições estabelecidas na premissa e os valores encontrados na realidade e C, definido em Z, o conjunto nebuloso presente na conclusão da regra. Então, para verificarmos o grau com que a premissa implica a conclusão, dados os valores encontrados na realidade, verificaremos o quanto α implica $\mu_C(z)$, verificando $I(\alpha, \mu_C(z))$ para todo $z \in Z$. A tabela 3.2 traz as principais operações de implicação encontradas na literatura.

Exemplo 2.4 [BNW96]:

Seja X um conjunto dos números reais entre 0 e 10, que será chamado de universo de discurso. Seja A um subconjunto nebulosos de X, constituída de todos os números reais entre 5 e 8 aproximadamente: A = [5,8]

Seja B um subconjunto nebuloso de X, constituída de todos os números reais aproximadamente 4: B = [4]

Estos subconjuntos podem ser observados nas figuras 3.4 e 3.5:

A figura 3.6 mostra a *intersecção* dos conjuntos nebulosos A e B.

A figura 3.7 mostra a *união* dos conjuntos nebulosos A e B.

Finalmente, a figura 3.8 mostra a *negação* do conjunto nebuloso A.

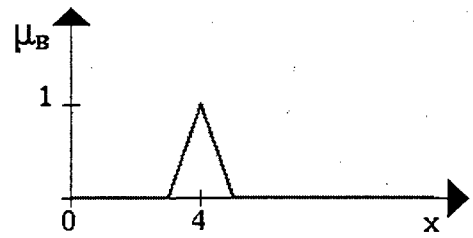


Figura 3.5: Número nebuloso ao redor de 4.

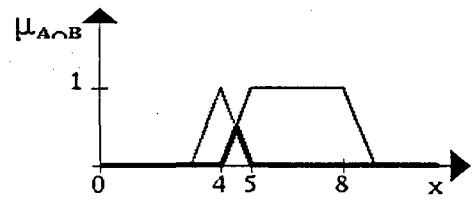


Figura 3.6: Intersecção dos conjuntos A e B.

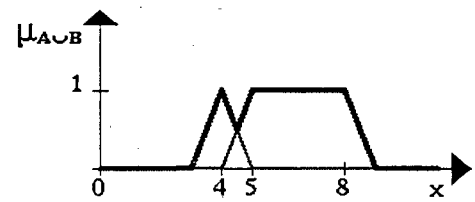


Figura 3.7: União dos conjuntos A e B.

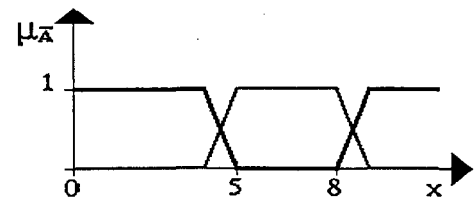


Figura 3.8: Negação do conjunto nebuloso A.

3.5 Regras de Inferência

A inferência nebulosa é um método de decisão que usa a teoria nebulosa para expressar o conhecimento humano na forma de regras lingüísticas. As regras de controle nebuloso são usualmente do tipo:

$$R_j : \text{Se } x_1 \text{ é } A_{1j} \text{ e } \cdots \text{ e } x_n \text{ é } A_{nj} \text{ então } y \text{ é } B_j.$$

onde os A_{ij} e B_j são conjuntos nebulosos, e a implicação é implementada como uma função de implicação nebulosa. Em cada ciclo do processo, os valores das variáveis x_{ij} são medidos e então comparados aos conjuntos nebulosos A_{ij} nas regras, gerando uma medida da adequação dos valores medidos à premissa da regra (utilizando uma T-norma para implementar o conectivo "e" na premissa das regras). A implicação então utiliza esta medida de adequação e o conjunto nebuloso B_j na conclusão para obter um valor B'_j para a variável de controle, em relação àquela regra. Os valores B'_j são então agregados em uma única ação de controle C, utilizando uma T-norma ou T-conorma. Em alguns modelos os B'_j e C são nebulosos, e então é necessário se determinar um valor preciso para a variável de controle, a partir de C [San97].

Os especialistas podem expressar até mesmo sistemas mais complexos em forma de regras nebulosas e funções de pertinência. As regras são denominadas nebulosas porque não são comandos na forma de sinais, mas na forma de ações, as quais definem diferentes níveis de ações; tais como *alto*, *baixo*, etc.

Exemplo 2.5 [YZ91]:

Se a TEMPERATURA é *alta* **então** a PRESSÃO é *baixa*.

A declaração condicional está formada por duas partes: TEMPERATURA é *alta* e PRESSÃO é *baixa*, unidas pela palavra condicional **se** e a implicação **então**. Cada parte é chamada de predicado nebuloso. A primeira sentença "TEMPERATURA é *alta*" é o antecedente; e a segunda sentença "PRESSÃO é *baixa*" é a conclusão ou ação [YZ91].

Um mecanismo de inferência é um procedimento para computar um resultado, que pertence à conclusão ou conseqüência, quando são definidos uma série de fatos que pertencem ao antecedente. Este procedimento de inferência utiliza a declaração condicional **se ... então**. Então, o mecanismo de inferência é uma forma de raciocinar.

Deve ser observado que os termos lingüísticos são aproximadas e vagas, portanto, a conseqüência será uma aproximação lingüística ao tópico de interesse. Isto quer dizer que nem sempre é possível afirmar que a conclusão é uma decisão exata, mas que é a melhor possível dentro do intervalo de ambigüidade dos fatos e da experiência prévia relacionada com esses fatos [Zad65]. Assim, os mecanismos de inferência são, portanto, mecanismos de

raciocinar aproximados. Entre os critérios para escolher um bom mecanismo de inferência destacam-se: a melhor aproximação e a menor carga computacional [YZ91].

3.6 Controle Nebuloso

Em muitos casos o controle automático de um processo complexo, onde não se conhece com precisão sua dinâmica, torna-se menos eficiente do que o controle realizado por um operador humano experimentado. Assim, um controlador nebuloso executa o controle através de um algoritmo baseado na experiência de um operador humano usando-se um conjunto de regras lingüísticas que descrevem a estratégia de controle utilizada. Estas variáveis lingüísticas são descritas por conjuntos nebulosos.

Por exemplo, o conjunto de regras de decisão lingüística tem a seguinte forma:
"se o incremento na temperatura é grande", então "diminuir bastante a pressão".

Definindo-se os conjuntos nebulosos para cada caso e transformando-se as regras em implicações nebulosas do tipo: *se* A *então* B, a estratégia de controle humana pode ser convertida em um algoritmo de controle e implementada em um computador.

Cabe ressaltar que na literatura são conhecidos dois tipos de controladores, aqueles que representam as regras com ações ou conclusões *simbólicas* e aqueles que têm conclusões *numéricas* de forma direta. O controlador de Mamdami corresponde ao primeiro tipo, e o de Sugeno ao segundo.

Formalmente as regras do controlador de Mamdani têm a seguinte estrutura [MA74]:

$$R_j : \text{Se } x_1 \text{ é } A_{1j} \text{ e } \cdots \text{ e } x_n \text{ é } A_{nj} \text{ então } y \text{ é } B_j.$$

Os termos desta expressão foram definidos na seção 3.5.

A conclusão deste tipo de regras é simbólica, por exemplo: "girar à direita". Esta ação deve ser desnebulisada para se obter uma saída numérica a ser aplicada ao processo em questão.

Por outro lado, as regras do controlador de Sugeno podem ser representadas como [TS83]:

$$R_j : \text{Se } x_1 \text{ é } A_{1j} \text{ e } \cdots \text{ e } x_n \text{ é } A_{nj} \text{ então } y \text{ é } f^j(x_1, ..., x_nj).$$

onde os A_{ij} e B_j são conjuntos nebulosos, e a implicação é implementada como uma função de implicação nebulosa. O termo $f^j(x_1, ..., x_nj)$ constitui a principal diferença em relação à estrutura das regras do controlador de Mamdani. Este termo representa que a conclusão de cada regra é uma função f^j das entradas do controlador. Funções lineares com coeficientes constantes simplificam este tipo de regras.

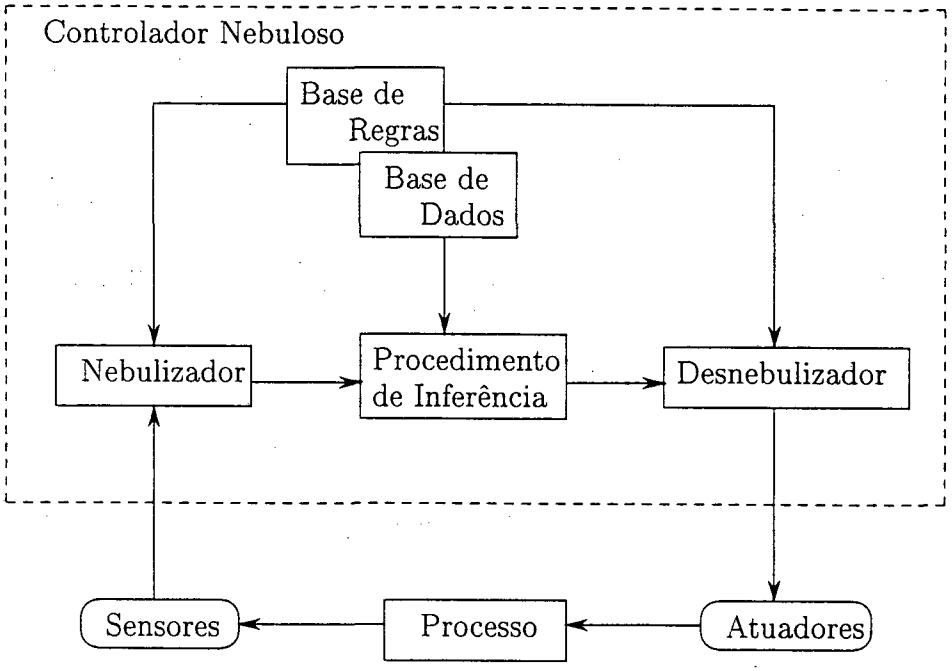


Figura 3.9: Estrutura do controlador nebuloso.

3.6.1 Partes básicas de um controlador nebuloso

O controlador nebuloso é formado por três partes básicas: o nebulizador, o mecanismo de inferência, e o desnebulizador [JVB94](figura 3.9).

Deve-se observar que tanto as leituras dos sensores quanto os sinais enviados aos atuadores do sistema de controle não são nebulosos, e portanto são necessários elementos adicionais entre o controlador nebuloso e o processo a ser controlado. Estes elementos são denominados nebulizador e desnebulizador, e estão posicionados na entrada e saída do sistema de controle, respectivamente.

No bloco nebulizador são recebidos os valores das variáveis medidas. Então, através de um mapeamento o valor da variável é transferido ao correspondente universo de discurso. Em seguida é realizada a nebulização. Existem diversos métodos de nebulização: por exemplo, a geração de um *singleton*, no qual a função de pertinência é zero exceto no ponto dado recebido. Algumas funções de pertinência típicas são as triangulares, trapezoidais e as Gausianas [God97].

Por outro lado, as regras de controle são construídas no bloco de inferência. Existem diferentes versões para realizar a *implicação* nebulosa (ver seção 3.4).

Depois de passar através dos dois blocos anteriores, a ação ou conclusão é desnebulizada para obter um valor que constitui o comando de controle do processo. Neste bloco,

destacam-se os seguintes métodos: o máximo, o centróide, o centro de área, entre outros [JVB94].

3.6.2 Exemplo: Pêndulo invertido

Este exemplo é citado com muita freqüência na literatura [BNW96]. O problema consiste em equilibrar um pêndulo numa plataforma móvel, a qual pode movimentar-se em duas direções: a esquerda ou a direita (figura 3.10).

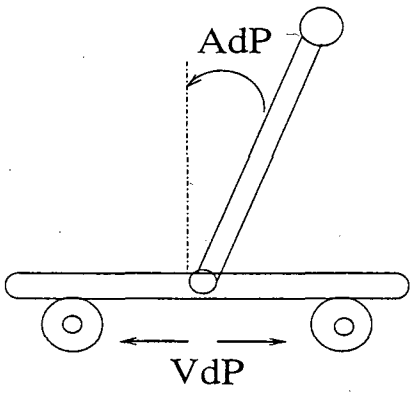


Figura 3.10: Pêndulo invertido (AdP: Ângulo do pêndulo e VdP: Velocidade da plataforma).

As variáveis lingüísticas são: o *ângulo* entre a plataforma e o pêndulo e sua *velocidade angular* (entradas); e a *velocidade* da plataforma, que é a saída do sistema. São definidas as seguintes funções de pertinência para as variáveis lingüísticas (figuras 3.11 - 3.13):

- Negativo Grande
- Negativo Pequeno
- Zero
- Positivo Pequeno
- Positivo Grande

Considera-se que a posição inicial do pêndulo está na direção vertical superior, e que não são atingidos ângulos superiores a 45^o em nenhuma direção.

Existem várias propostas. A seguir é definido um conjunto de regras de inferência. Por exemplo, considere que o pêndulo está na posição superior, onde o ângulo é zero, e que não está se movendo. Nesta situação não precisa ser exercida nenhuma ação de controle e, portanto, a velocidade da plataforma é zero.

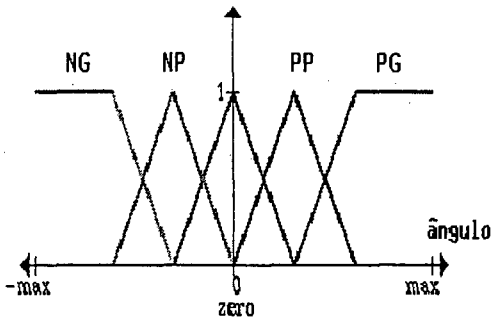


Figura 3.11: Ângulo do pêndulo.

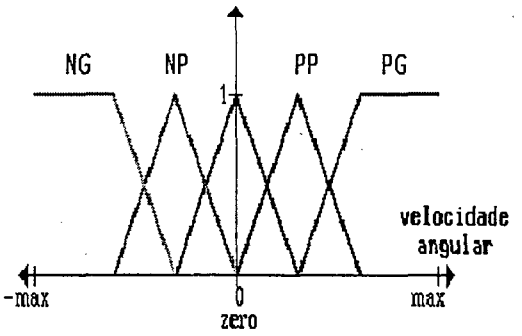


Figura 3.12: Velocidade angular do pêndulo.

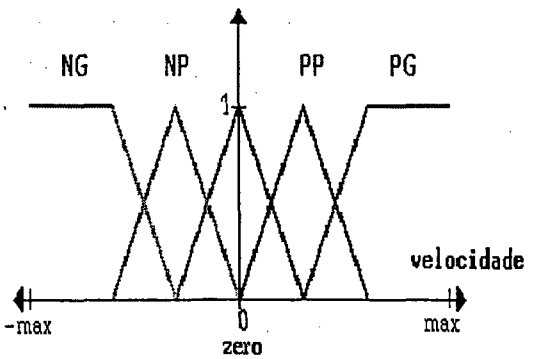


Figura 3.13: Velocidade da plataforma.

Tabela 3.3: Regras de inferência da velocidade da plataforma (saída) para o problema do pêndulo invertido

Veloc. Angular	Ângulo				
	NG	NP	Z	PP	PG
NG			NG		
NP			NP	Z	
Z	NG	NP	Z	PP	PG
PP		Z	PP		
PG			PG		

Por outro lado, se o pêndulo está na posição superior, mas está se movendo com velocidade baixa na direção considerada positiva, então, deve ser aplicada à plataforma uma velocidade linear baixa na mesma direção (positiva) para compensar o movimento do pêndulo. Assim, sucessivamente é elaborado o conjunto de regras.

Depois estas regras são colocadas na forma se ... então. Por exemplo, as regras mencionadas podem ser formalizadas como:

Se o ângulo é zero *e* a velocidade angular é zero, *então*, a velocidade da plataforma é zero;

Se o ângulo é zero *e* a velocidade angular é positiva baixa, *então*, a velocidade da plataforma é positiva baixa.

Na tabela 3.3 é mostrado o conjunto de regras aplicadas neste problema.

Observar que NG é a abreviação de Negativo Grande, NP é a abreviação de Negativo Pequeno e Z é a abreviação de Zero. De forma similar, PG é a abreviação de Positivo Grande e PP é a abreviação de Positivo Pequeno. Os espaços em branco representam combinações que não existem [BNW96].

Na continuação, será mostrado todo o processo de elaboração de um comando de controle a partir dos valores das variáveis de entrada. Serão assumidos valores para o ângulo e a velocidade angular do pêndulo (entradas).

a) nebulização

A figura 3.14 mostra a função característica da variável de entrada ângulo e a nebulização do valor atual do ângulo.

A figura 3.15 mostra a função característica da variável de entrada velocidade angular, e a nebulização do valor atual da velocidade.

b) mecanismo de inferência

Na seqüência, são aplicadas as regras de inferência. Observar que considerando os

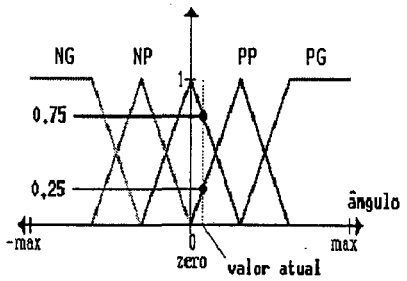


Figura 3.14: Nebulização do ângulo.

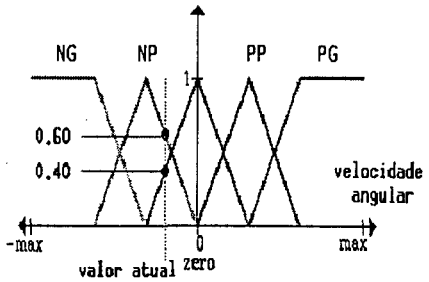


Figura 3.15: Nebulização da velocidade angular.

dados atuais das entradas somente são aplicadas quatro regras.

Por exemplo, vamos aplicar a seguinte regra para os dados atuais das variáveis de entrada (ver figura 3.16):

1) *Se* o ângulo é zero *e* a velocidade angular é zero, *então*, a velocidade é zero.

Observar que devido ao fato de que as duas partes da condição desta regra estão ligadas pelo conetivo *e*, é calculado o min(0.75, 0.4)= 0.4, o qual corresponde a intersecção destes valores no método de Mandani. Existem outras opções. Depois, observando-se a conclusão da regra, a saída é *zero*. É feito o mínimo do valor de compatibilidade da premissa de regras em sua conclusão.

Assim por diante, este processo é realizado para as restantes regras.

Na figura 3.17 mostra-se a aplicação da regra 2:

2) *Se* o ângulo é zero *e* a velocidade angular é negativa pequena, *então*, a velocidade da plataforma é negativa pequena.

é calculado o min(0.75, 0.6)= 0.6 e a conclusão da regra é *negativa pequena.*

Na figura 3.18 mostra-se a aplicação da regra 3:

3) *Se* o ângulo é positivo pequeno *e* a velocidade angular é zero, *então*, a velocidade da plataforma é positiva pequena.

É calculado o min(0.25, 0.4)= 0.25 e a conclusão da regra é *positiva pequena.*

Na figura 3.19, mostra-se a aplicação da regra 4:

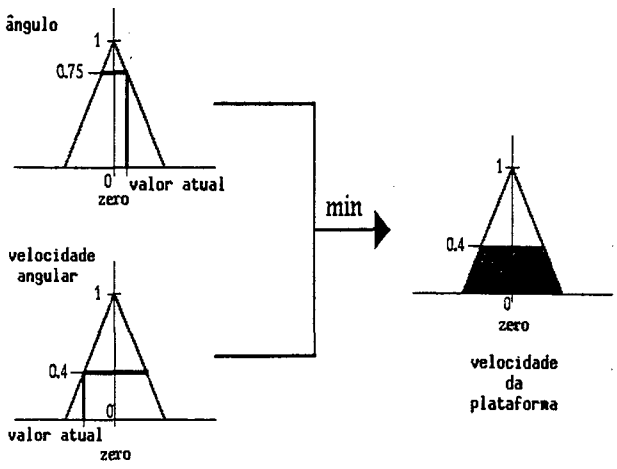


Figura 3.16: Aplicação da Regra 1.

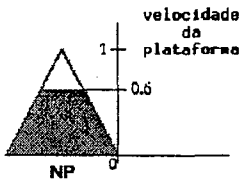


Figura 3.17: Aplicação da Regra 2.

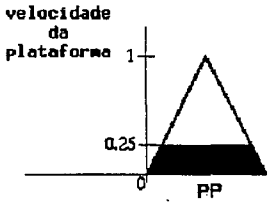


Figura 3.18: Aplicação da Regra 3.

4) *Se* o ângulo é positivo pequeno *e* a velocidade angular é negativa pequena, *então*, a velocidade da plataforma é zero.

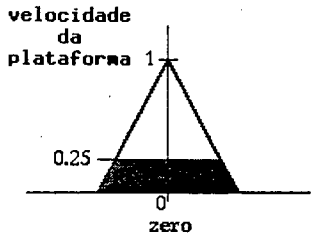


Figura 3.19: Aplicação da Regra 4.

É calculado o min(0.25, 0.6)= 0.25 e a conclusão da regra é *zero*.

A figura 3.20 mostra a união destas quatros regras.

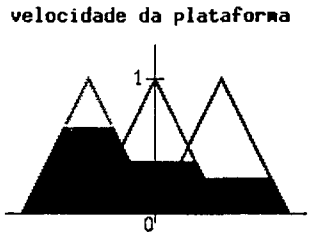


Figura 3.20: Resultado da aplicação das quatro regras.

c) desnebulização

A escolha do valor representativo da velocidade (desnebulização) pode ser realizada através de diferentes métodos. Neste exemplo foi aplicado o de o *centro de gravidade* ao conjunto nebuloso resultante das operações anteriores. A figura 3.21 mostra o resultado.

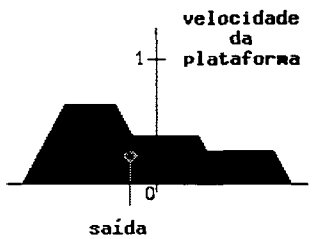


Figura 3.21: Desnebulização.

3.7 Aplicações

Alguns exemplos da aplicação da lógica nebulosa são [BNW96]:

- Controle de robôs (Hirota, Fuji Electric, Toshiba, Omron);

- Cruise-control para automóveis (Nissan, Subaru);
- Otimização em aplicações industriais (Omron, Mitsubishi, Aptronix);
- Produção de semicondutores (Canon);
- Sistema predictivo para a detecção de terremotos (Ins. of Sismology Bureau, Japan);
- Combinação de Redes Neurais e Logica Nebulosa (Matsushita);
- Diagnóstico de doenças (Kawasaki Medical School);
- Reconhecimento de padrões (CSK, Hitachi);
- Vôo assistido para helicópteros (Sugeno);
- Controle de elevadores (Fujitec, Hitachi, Toshiba);
- Melhoramento da segurança em reatores nucleares (Hitachi, Bernard).

Na literatura são abordadas outras aplicações [YZ91], [BNW96], etc., nas quais é possível observar que a lógica nebulosa tem se desenvolvido como uma ferramenta valiosa para o controle tanto de sistemas de baixo nível quanto de complexos processos industriais. Resulta interessante ressaltar que, embora criada nos Estados Unidos de América, a lógica nebulosa tem experimentado um rápido crescimento no Japão, onde o número de patentes aumenta de forma exponencial [JVB94].

3.8 Conclusão

A lógica nebulosa tem se tornado popular na comunidade de controle. A proliferação desta estratégia de controle adquire destaque pela introdução em produtos da indústria eletrônica japonesa, embora atualmente esta técnica tenha se estendido a outros campos. Processos de produção mais modernos requerem um controle mais avançado, onde mudanças freqüentes na taxa de produção, mistura de produtos, procedimentos, entre outros, dá lugar a uma situação totalmente diferente, onde uma linearização ao redor de condições operacionais é dificilmente possível [JVB94]. Neste ponto, resulta interessante a introdução de novas metodologias de controle, em particular do controle nebuloso.

Neste capítulo foram abordados os aspectos mais importantes relacionados à lógica nebulosa, tais como: os conjuntos nebulosos, as operações entre conjuntos, as regras de inferências, as etapas do controle segundo esta metodologia, bem como, algumas aplicações. Posteriormente, no Capítulo 6, será realizada uma aplicação desta técnica em robótica móvel.

Capítulo 4

Algoritmos Genéticos

"While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to especulate on new search points with expected improved performance."

David E. Goldberg [Gol89].

4.1 Introdução

Toda tarefa de busca e otimização possui vários componentes, entre eles: o espaço de busca, onde são consideradas todas as possibilidades de solução de um determinado problema, e a função de avaliação (ou função de custo), a qual é uma maneira de avaliar os membros do espaço de busca.

As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado utilizando ferramentas heurísticas diretamente associadas ao problema a ser solucionado. Geralmente, estes processos heurísticos não são algorítmicos e sua simulação em computadores pode ser muito complexa. Apesar destes métodos não serem suficientemente robustos, isto não implica que sejam inúteis. Na prática, eles são amplamente utilizados, com sucesso, em inúmeras aplicações.

Neste capítulo serão analisados os principais aspectos e características dos Algoritmos Genéticos (AGs) que os fazem extremamente interessantes como ferramenta de busca e otimização na solução de diferentes tipos de problemas [ANd96], [Coe97].

4.2 O que diferencia os AGs de outros métodos?

Os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização principalmente nos seguintes aspectos [ANd96]:

1. AGs trabalham com uma codificação do conjunto de parâmetros, e não com os próprios parâmetros (por exemplo, a representação binária dos cromossomos);
2. AGs trabalham com uma população, e não com um ponto isolado;
3. AGs utilizam informações de custo ou recompensa, e não derivadas, ou outro conhecimento auxiliar;
4. AGs utilizam regras de transição probabilísticas, e não determinísticas [Gol89];

Os Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais.

Cabe ressaltar que os AGs, além de serem uma estratégia de gerar e testar muito elegante, por serem baseados na teoria da evolução natural, são capazes de identificar e explorar fatores ambientais e convergir para soluções ótimas, ou aproximadamente ótimas, em níveis globais.

Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes. Este é o conceito básico da teoria da evolução natural. Assim, a área biológica mais proximamente ligada aos Algoritmos Genéticos é a Genética Populacional.

4.3 Conceitos básicos

Antes de prosseguir com a análise das características destes algoritmos, são necessários alguns conceitos básicos (ver [Coe97]).

- fenótipo: é o conjunto de características fornecidas pelo genótipo;
- genótipo: consiste de um arranjo dos genes;
- indivíduo: exemplar de uma espécie que interage com o meio ambiente;
- espécie: grupo de populações com genótipos semelhantes;
- população: é o conjunto de indivíduos da mesma espécie;

- cromossomo: componente genético responsável pelo fenótipo do indivíduo;
- genes: são a unidade básica do cromossomo, que definem, de acordo com seu valor e posição, uma característica do cromossomo;
- reprodução: multiplicação de novos indivíduos a partir de outros da mesma espécie;

Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Logo, durante o processo evolutivo, esta população é avaliada e, para cada indivíduo, é dada uma nota, ou índice, refletindo sua habilidade de adaptação a determinado ambiente. Uma porcentagem dos mais adaptados são mantidos, enquanto os outros são descartados (darwinismo). Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais através de mutações e cruzamento (crossover), ou recombinação genética, gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada [ANd96].

Embora possam parecer simplistas do ponto de vista biológico, estes algoritmos são suficientemente complexos para fornecer mecanismos de busca adaptativos poderosos e robustos.

4.4 Características Gerais dos AGs

Os Algoritmos Genéticos são algoritmos de otimização global baseados nos mecanismos de seleção natural e da genética. Eles empregam uma estratégia de busca paralela e estruturada mas aleatória, que é voltada em direção ao reforço da busca de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Apesar de aleatórios, eles não são caminhadas aleatórias não direcionadas, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos. Isto é feito através de processos iterativos onde cada iteração é chamada de geração.

Durante cada iteração os princípios de seleção e reprodução são aplicados a uma população de candidatos que pode variar dependendo da complexidade do problema e dos recursos computacionais disponíveis. Através da seleção, se determina quais indivíduos conseguirão reproduzir-se, gerando um número determinado de descendentes para a próxima geração com uma probabilidade determinada pelo seu índice de aptidão. Em outras palavras, os indivíduos com maior adaptação relativa têm maiores chances de se reproduzir [ANd96].

O ponto de partida para a utilização de Algoritmos Genéticos como ferramenta para solução de problemas é a representação destes problemas de maneira que os AGs possam trabalhar adequadamente sobre eles. Assim, a maioria das representações são genotípicas, as quais utilizam vetores de tamanho finito em um alfabeto também finito.

Tradicionalmente, os indivíduos são representados genotipicamente por vetores binários, onde cada elemento de um vetor denota a presença (1), ou ausência (0) de uma determinada característica: o seu genótipo. Estes elementos são combinados para formar as características de adaptabilidade do elemento, calculada através das função que determina o índice de aptidão. Estas características são chamadas de fenótipo do indivíduo. Teoricamente, esta representação é independente do problema, pois uma vez encontrada a representação em vetores binários, as operações padrão podem ser utilizadas, facilitando seu emprego em diferentes classes de problemas [ANd96].

Também a utilização de representações em níveis de abstração mais altos tem sido investigada. Como estas representações são mais fenotípicas, facilitariam sua utilização em determinados ambientes onde essa transformação "fenótipo - genótipo" é muito complexa. Neste caso, precisam ser criados os operadores específicos para utilizar estas representações.

4.5 Algoritmo

O princípio básico do funcionamento dos AGs é que um critério de seleção vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos. A maioria dos métodos de seleção são projetados para escolher, preferencialmente, indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população. Um método de seleção muito utilizado é o Método da Roleta, onde indivíduos de uma geração são escolhidos para fazer parte da próxima geração através de um sorteio de roleta [Bit98a].

Neste método, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão (ver figura 4.1). Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor da roleta. Finalmente, a roleta é girada um determinado número de vezes (dependendo do tamanho da população) e são escolhidos como indivíduos que participarão da próxima geração, aqueles sorteados na roleta [Gol89].

Um conjunto de operações é necessário para que, dada uma população, se consiga gerar populações sucessivas que (espera-se) melhorem sua aptidão com o tempo. Estes operadores são: cruzamento (crossover) e mutação. Eles são utilizados para assegurar

	Indivíduo	Aptidão f(Si)	Aptidão Relativa
S_1	01101	169	14.4
S_2	11000	576	49.2
S_3	01000	64	5.5
S_4	10011	361	30.9
Total		1170	100.0

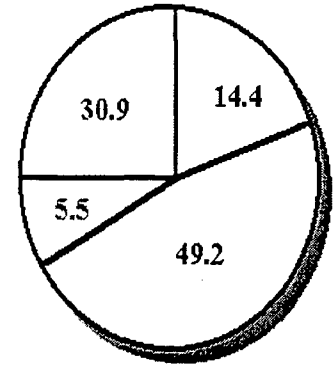


Figura 4.1: Indivíduos de uma população e a sua correspondente roleta de seleção.

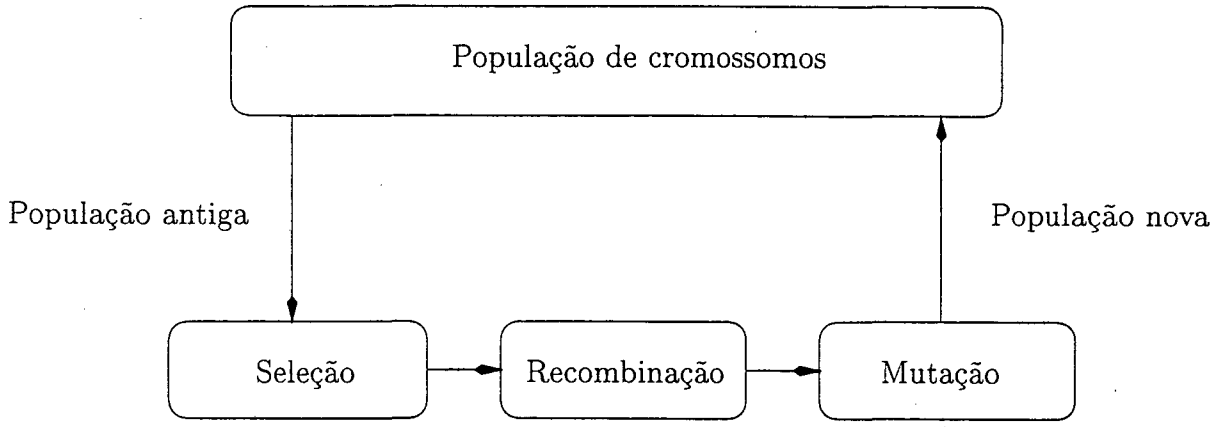


Figura 4.2: Representação do Algoritmo Genético.

que a nova geração, mesmo que totalmente diversa da anterior, possua, de alguma forma, características de seus pais; ou seja, a população diversifica-se, mas mantém as características de adaptação adquiridas pelas gerações anteriores. Também para prevenir que os melhores indivíduos não desapareçam da população pela manipulação dos operadores genéticos, eles podem ser automaticamente colocados na próxima geração através da reprodução elitista. Esse ciclo é repetido um determinado número de vezes.

Ver na subseção 2.3.2 um exemplo de algoritmo genético. Durante este processo, os melhores indivíduos, assim como alguns dados estatísticos, podem ser coletados e armazenados para avaliação.

Este procedimento está representado na figura 4.2 [Coe97].

Estes algoritmos, apesar de serem computacionalmente muito simples, são bastante poderosos. Além disso, não estão limitados por suposições sobre o espaço de buscas

```
Antes:  1 1 |1| 0 0

Depois: 1 1 |0| 0 0
```

Figura 4.3: Exemplo de mutação

relativas a continuidades, existências de derivadas, etc.

4.5.1 Operadores Genéticos

O princípio básico dos operadores genéticos é transformar a população através de sucessivas gerações estendendo a busca até chegar a um resultado satisfatório. Assim, os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores. Destacam-se os seguintes operadores:

- Operador de mutação:

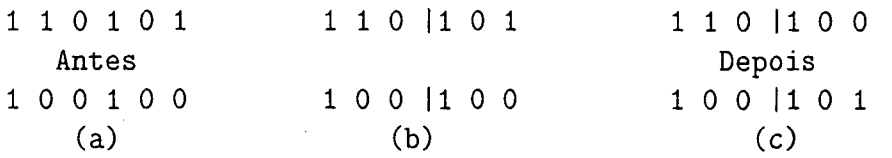
  O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população (figura 4.3). Ele altera arbitrariamente um ou mais componentes de uma estrutura escolhida, fornecendo assim meios para a introdução de novos elementos na população [ANd96]. Observa-se na figura 4.3 que foi realizada uma mutação no bit 3.

  A mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero, além de contornar o problema de mínimos locais pois com este mecanismo altera-se levemente a direção da busca. O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação P_m; geralmente se utiliza uma taxa de mutação pequena, pois é um operador genético secundário.

- Operador de recombinação (crossover):

  O cruzamento é o operador responsável pela recombinação das características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso é aplicado com probabilidade dada pela taxa de crossover P_c, que deve ser maior que a taxa de mutação P_m [ANd96].

  Este operador pode, ainda, ser utilizado de várias maneiras. As mais utilizadas são:



(a) dois indivíduos são escolhidos.
(b) um ponto de crossover é escolhido.
(c) são recombinadas as características, gerando dois novos indivíduos.

Figura 4.4: Exemplo de crossover de um ponto.

- um-ponto: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais são trocadas. As informações anteriores a este ponto em um dos pais são ligadas às informações posteriores à este ponto no outro pai. Observa-se na figura 4.4 que o cruzamento é realizado a partir do quarto bit.

- multi-pontos: é uma generalização da idéia de troca de material genético através de pontos, onde podem ser utilizados muitos pontos de cruzamento.

- uniforme: não utiliza pontos de cruzamento, mas determina através de um parâmetro global, qual a probabilidade de cada variável ser trocada entre os pais.

4.5.2 Parâmetros Genéticos

É importante analisar a maneira que alguns dos parâmetros influem no comportamento dos Algoritmos Genéticos, para que eles possam ser estabelecidos conforme as necessidades do problema e dos recursos disponíveis [ANd96].

Parâmetros Genéticos:

- Tamanho da População. O tamanho da população afeta o desempenho global e a eficiência dos AGs. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população, geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.

- Taxa de Cruzamento. Quanto maior for esta taxa, mais rapidamente as novas estruturas serão introduzidas na população. Mas se esta for muito alta, as estruturas com boas aptidões poderão ser retiradas mais rapidamente; assim, a maior parte da população será substituída e pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode-se tornar muito lento.

- Taxa de Mutação. Considerando que o operador de mutação desempenha um papel secundário (tanto nas populações naturais quanto artificiais), os valores desta taxa num algoritmo genético simples estão na ordem de 1/100; ou seja, acontece uma mutação por cada cem bits transferidos.

- Intervalo de Geração. Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode se tornar muito lento.

A literatura reporta que na grande maioria das aplicações, o AG com representação binária é utilizado com um tamanho de população entre 30 e 200, uma probabilidade de recombinação (P_c) entre 0.5 e 1.0 e uma probabilidade de mutação (P_m) entre 0.001 e 0.05 [Coe97]. Quanto às funções de avaliação, estas são independentes do AG.

4.6 Hibridização

A técnica de hibridização resulta na integração de uma boa maneira convencional de resolver um problema, aos conceitos usuais de AG's. O resultado costuma ser melhor que o obtido com qualquer uma das duas técnicas isoladamente [Dav91]. A hibridização agrega a representação usual de dados no domínio original, bem como as técnicas de otimização já existentes. Isto permite a incorporação de heurísticas otimizadoras ao conjunto de operadores genéticos (recombinação e mutação) que passam portanto a ser dependentes do domínio. Nesse sentido, o algoritmo genético passa a ser muito mais uma filosofia de otimização do que um método pronto para utilização.

Um exemplo de hibridização possível é quando o problema exige codificação com base em números reais e não em números binários. Alguns conceitos teriam que ser adaptados: por exemplo, a mutação não seria mais a troca simples de um "bit", mas a geração de um novo real, possivelmente dentro de um intervalo dado. Já a recombinação de dois reais poderia ser qualquer número compreendido entre eles, dado por uma média entre dois reais (aritmética, geometrica), simples ou ponderada dá sempre resultados entre os dois reais.

Outra possibilidade de hibridização é quando o problema envolve algoritmos de ordenamento como, por exemplo, os problemas do caixeiro viajante e da coloração de um grafo sem que nenhum par de nodos conectados tenham côr igual [Bit98b].

4.7 Aplicações

Um sistema com bom desempenho em um ambiente dinâmico, geralmente exige soluções adaptativas. Sistemas adaptativos tentam resolver problemas acumulando conhecimento sobre o problema e utilizando estas informações para gerar soluções aceitáveis. Estes problemas, tipicamente, encontram-se nas áreas de configuração de sistemas complexos, alocação de tarefas, seleção de rotas, e outros problemas de otimização e aprendizado de máquina. Assim, os AGs são exemplos de soluções para estes tipos de problemas de otimização.

A seguir são mencionados alguns exemplos de sistemas que usam esta técnica [ANd96], [Coe97]:

- Controle de Sistemas Dinâmicos;
- Indução e Otimização de Bases de Regras;
- Encontrar Novas Topologias Conexionistas:
- Engenharia de Sistemas Neurais Artificiais,
- Modelagem de Estruturas Neurais Biológicas,
- Simulação de Modelos Biológicos:
- Evolução Interativa de Imagens;
- Composição Musical.

4.8 Conclusão

Neste capítulo foram analisados os principais aspectos relacionados aos Algoritmos Genéticos. Foram definidos os conceitos básicos e os passos do algoritmo. Também foram mencionados alguns dos usos dos AGs, os quais mostram a crescente diversidade de aplicações.

No capítulo 6, será realizada uma aplicação desta técnica em robótica móvel.

Capítulo 5

Robôs Móveis Autônomos

" Robots are built to replace human beings in performing tasks that humans cannot or prefer not to do. They can visit dangerous areas in nuclear plants, transport goods, clean floors, search for mines, or harvest crops in fields."

Jelena Godjevac [God97].

5.1 Introdução

Um dos principais objetivos da Inteligência Artificial é o de aprofundar o conhecimento da inteligência natural mediante a geração e a análise de comportamentos artificiais. Neste sentido, os robôs móveis representam uma plataforma ideal para investigar os processos de percepção e ação autônomos e inteligentes; e portanto, são uma ferramenta muito útil para validar as teorias estabelecidas pela Inteligência Artificial.

Os primeiros robôs móveis foram produzidos para aplicações em locais perfeitamente estruturados, de forma que a incerteza sobre o conhecimento do mundo ficava reduzida a um nível mínimo. Isto porque quanto maior é o conhecimento sobre o ambiente de trabalho, menos necessárias serão as capacidades sensoriais e de raciocínio do robô. De fato, a grande maioria de robôs industriais trabalha num ambiente bem conhecido, onde apenas são necessários sensores externos, e onde apenas se precisa tomar decisões. O robô que foi produzido para aplicações em locais perfeitamente estruturados se limita a repetir uma determinada seqüência de movimentos.

Porém, nem todos os ambientes nos quais possa trabalhar um robô móvel são totalmente estruturados. Existem ambientes onde é muito difícil conhecer de antemão a localização exata de todos os elementos presentes no ambiente e planejar todas as possíveis ações que deve realizar o robô [Fon96]. Portanto, é necessário que o robô seja capaz de to-

mar decisões de maneira independente levando em consideração as diretrizes de um plano geral. Ou seja, o robô ideal seria aquele que realizasse as ações propostas em um plano proporcionado por um usuário, sendo capaz de atuar de forma autônoma ante eventos imprevistos.

O objetivo deste capítulo é apresentar brevemente os aspectos mais importantes relacionados com as arquiteturas e as estratégias de controle de robôs móveis autônomos. Especial atenção será dada à definição de duas tarefas básicas de navegação: evitar obstáculos ("Avoid") e seguir paredes ("Follow Wall").

5.2 Evolução das Arquiteturas de Robôs Móveis Autônomos

Inicialmente os investigadores trataram de conectar os manipuladores de símbolos, que tinham sido desenvolvidos no campo da Inteligência Artificial, aos sensores e atuadores dos robôs móveis. Um dos principais objetivos era o de gerar um modelo simbólico do mundo (no espaço cartesiano) a partir da informação sensorial de forma que a máquina de inferência pudesse raciocinar e planejar de antemão os movimentos do robô. Primeiro, foi especificado o formato da base simbólica de conhecimento e a máquina de inferência, e mais tarde foram desenvolvidos os módulos de percepção que deveriam prover os símbolos necessários. Por exemplo, o engenheiro de sistemas desenvolve a base de conhecimento e a máquina de inferência do robô e supõe que o engenheiro de "visão artificial" realizará um módulo que transforme as imagens percebidas em símbolos. Em geral, a proposta da Inteligência Artificial clássica era a de realizar uma planificação de longo prazo baseada no modelo do mundo em um instante dado. Se o robô encontrava-se ante um evento imprevisto, o sistema de planejamento realizava um novo plano de atuação.

5.2.1 Sistemas Distribuídos

Em meados dos anos oitenta, Brooks ([Bro86]), em resposta a esta visão tradicional da Inteligência Artificial aplicada à robótica móvel, iniciou uma nova filosofia de projeto que ele chamou de Sistemas Reativos. Ele substituiu a modularização funcional por módulos geradores de comportamentos, ou agentes. Uma hierarquia de processos toma conta do sistema, onde cada um destes processos é extremadamente simples e dotado de uma funcionalidade completa (*evitar obstáculos*, *seguir uma parede*, etc.). O curto caminho entre sensores e atuadores (do ponto de vista computacional) proporcionava uma resposta em tempo real do robô ante mudanças no ambiente. A implementação do sistema

realiza-se de baixo para cima, onde primeiro trata-se da segurança do robô mediante comportamentos básicos como *evitar obstáculos*, e depois são acrescentadas novas e mais complexas as atitudes; como a confecção de mapas do ambiente, ou a manipulação de objetos [Fon96], etc.

Tanto as arquiteturas reativas, como as baseadas em comportamentos, estão baseadas na ação imediata ante o estímulo percebido. Os objetivos do sistema são variados e as condições do ambiente ditam qual deles é o prioritário.

Os robôs projetados mediante este tipo de técnica apresentam comportamentos similares aos insetos [Bro91], e são utilizados como ferramentas de estudo no campo da Vida Artificial, que trata de emular os processos básicos da vida através de computadores. Da mesma forma, as arquiteturas multiagentes são amplamente utilizadas em outros campos da Inteligência Artificial, tais como em ambientes complexos, ferramentas de buscar informações na Internet, ou na cooperação entre vários robôs móveis [CB98], etc.

5.3 Estrutura do sistema de controle de um robô móvel

Na figura 5.1, observa-se a composição do sistema de controle de um robô móvel, a qual está dividida em três partes: os agentes motores de navegação, os agentes motores de visão e o planejamento [Fon96].

- Agentes Motores de Navegação: são capazes de interpretar a informação sensorial fornecida pelos sensores (ultrasom, laser, infravermelhos, etc.) com os quais está equipado o robô para realizar tarefas de navegação de forma autônoma. Em geral, devido aos numerosos erros que são cometidos nas leituras dos sensores, é gerada uma representação local do ambiente que permite eliminar o ruído e as informações imprecisas do meio. Esta representação se constrói baseada na informação dos sensores.

- Agentes Motores de Visão: são especialmente projetados para participar nas tarefas de procurar, detectar, seguir e reconhecer a localização de marcas no ambiente (por exemplo, um conjunto de balizas formadas por uma combinação de cores). O sistema de visão artificial funciona em paralelo com o sistema de navegação, ou seja, o sistema é capaz de seguir um objeto determinado ao mesmo tempo que são evitados os obstáculos imprevistos na trajetória do robô.

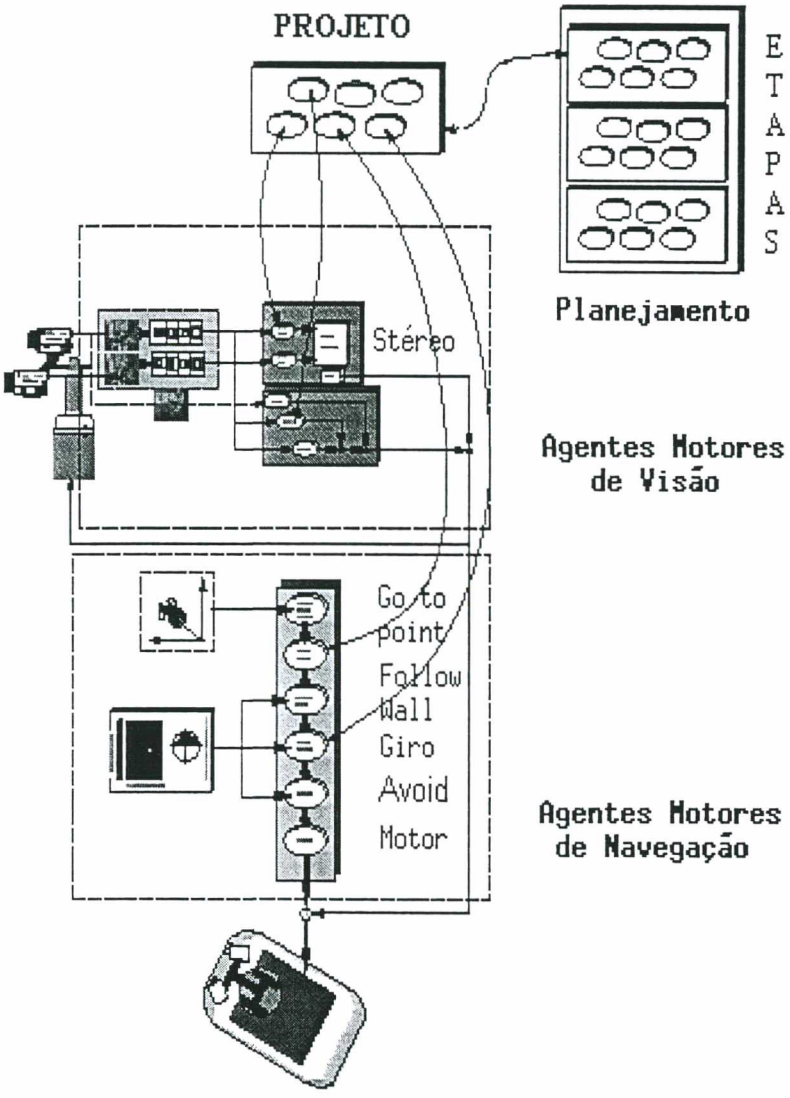


Figura 5.1: Estrutura do sistema de controle de um robô móvel [Fon96].

- Planejamento: o planejamento das atividades de um robô móvel implica a necessidade de indicar explicitamente, e de forma flexível, um conjunto de metas ao robô. Em geral, o plano é formado por um conjunto de passos, onde cada passo é formado por um objetivo e um conjunto de recomendações [Fon96]. Tanto as metas, quanto as recomendações de cada passo, são baseadas nas capacidades de ação e percepção do robô móvel. Neste sentido, o robô é capaz de evitar obstáculos, seguir paredes e ir a uma posição determinada. Da mesma forma, é capaz de buscar, reconhecer, seguir e estimar a posição em três dimensões de um conjunto de balizas visuais. Para que o planejamento das ações não seja um processo independente da percepção e ação imediata, todas estas habilidades do robô são implementadas como um conjunto de agentes independentes que atuam em paralelo.

Este trabalho tem um interesse particular em estudar o trabalho dos agentes motores de navegação. Isto será tratado na subseção seguinte.

5.3.1 Agentes motores de navegação

Nesta seção são tratados os aspectos do projeto de um sistema capaz de interpretar a informação sensorial proporcionada pelos sensores, com os quais está equipado um robô móvel, para poder realizar um conjunto de tarefas básicas. Serão tratadas duas tarefas de navegação: *evitar obstáculos* e *seguir paredes*.

Primeiramente serão definidos alguns conceitos básicos.

5.3.1.1 Sistemas de coordenadas

O sistema de coordenadas permite localizar o robô no ambiente. Na figura 5.2 é possível observar um exemplo de sistema de coordenadas de um robô móvel. Observa-se que neste caso ela está centrada no eixo que une as rodas motrizes. Este sistema de coordenadas é solidário ao robô, no sentido de que o eixo de ordenadas sempre tem a mesma direção e sentido que o movimento do robô, sendo que, por exemplo, $\Theta = 90^o$ corresponde à direção frontal.

5.3.1.2 Etiquetas e Intervalos

Para *evitar obstáculos*, ou *seguir paredes*, são necessárias abstrações de alto nível mediante a definição de etiquetas como *frente*, *atrás*, *esquerda* ou *direita*. A tabela 5.1 mostra um exemplo da relação entre os intervalos de domínio de Θ e a etiqueta associada a cada intervalo.

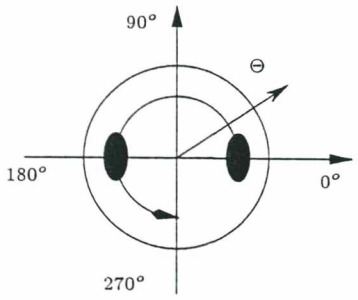


Figura 5.2: Sistema de referência de um robô móvel.

Tabela 5.1: Etiquetas que podem ser estabelecidas em função do intervalo de domínio de Θ.

Intervalo de Dominio	Etiqueta
Θ [90°, 270°]	Esquerda
Θ [270°, 90°]	Direita
Θ [0°, 180°]	Frente
Θ [180°, 0°]	Atrás

5.4 Evitar obstáculos ("Avoid")

O comportamento *evitar obstáculos* recebe as instruções de controle dos níveis superiores e, mediante o estudo da distância que separa o robô dos obstáculos que o rodeiam, interpreta as instruções de controle, hierarquicamente, modula as instruções recebidas e dá como saída uma instrução de controle próprio em termos de ações de controle a serem realizadas pelos atuadores do robô móvel. "Avoid" tentará aproximar-se o mais possível dos níveis superiores. Em geral, a instrução que recebe "Avoid" é um vetor (velocidade, giro); onde a velocidade é a velocidade linear (por exemplo, em cm/s) que deve seguir o robô e giro é o giro a realizar (em rad). A saída de *evitar obstáculos* é também um vetor (velocidade, giro) que alimenta os agentes de mais baixo nível, ou diretamente, o agente atuador que é quem transforma a saída de "Avoid" em instruções que os controladores do robô móvel possam entender (figura 5.3) [Fon96].

5.4.1 Entrada e saída

Em termos computacionais, o agente "Avoid" recebe um vetor de entrada (velocidade, giro) e mediante uma avaliação dará como resultado um vetor de saída (velocidade, giro).

Figura 5.3: Ações de controle e instruções de entrada e saída em *evitar obstáculos.*

Portanto,

Seja *de* e *ds* respectivamente os valores de entrada e saída. O agente *evitar obstáculos* pode ser definido como uma função A, tal que $ds = A(de)$.

As próprias características do robô influem na hora de desenvolver os algoritmos de controle do agente *evitar obstáculos.* Assim, é desejável que os comandos de controle do robô admitiam um intervalo de velocidades contínuo, mais isto nem sempre é possível. Em um robô onde o intervalo de velocidades é suave e contínuo, é possível implementar um controle de giro e de velocidade como uma função linear da instrução de entrada.

O comportamento de evitar obstáculos relaciona uma série de etiquetas lingüísticas com os diferentes intervalos do domínio da função de avaliação A, para definir as direções esquerda, direita, frente e atrás (ver seção 5.3.1.2).

5.4.2 Controle

O procedimento de decisão da ação de controle a realizar pelo agente *evitar obstáculos* ("Avoid"), pode-se resumir como [Fon96]:

1. Verificar se é possível realizar o giro estabelecido pela instrução de entrada;
2. Quando esta ação de giro não é possível, procurar o giro mais próximo ao desejado;
3. Estabelecer a velocidade de giro e a velocidade linear, dependendo da proximidade dos obstáculos do ambiente.

5.5 Seguir paredes ("Follow Wall")

O objetivo do comportamento *seguir paredes* é alinhar o robô móvel com uma parede a partir da informação oferecida pelos módulos sensoriais. A ação de controle é geralmente uma ação de *giro a realizar* (ver figura 5.4) [Fon96].

Figura 5.4: Ações de controle e instruções de entrada e saída em *seguir paredes*.

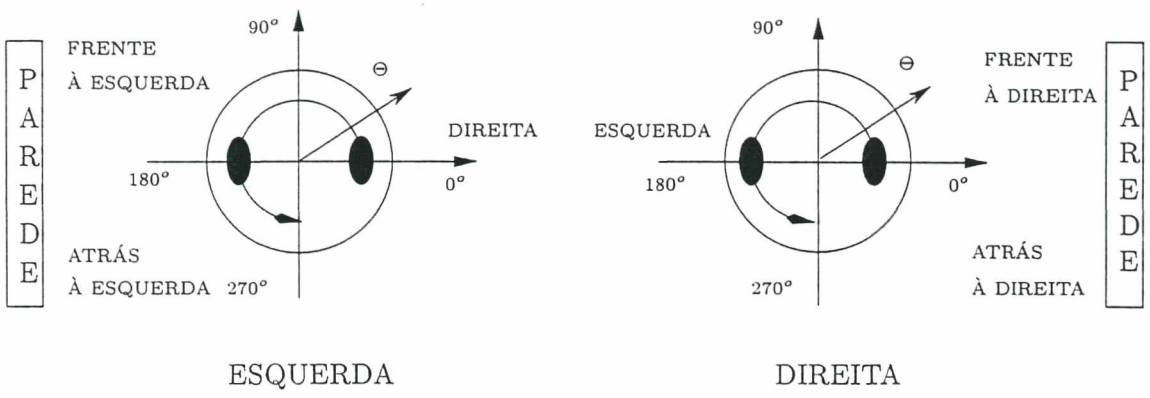


Figura 5.5: Intervalo de valores de Θ utilizados pelo agente *seguir paredes*.

5.5.1 Entrada e saída

O vetor de entrada é do tipo (velocidade, giro) por motivos de compatibilidade com o agentes *evitar obstáculos*. O agente *seguir paredes* ("Follow Wall") receberá um vetor de entrada (velocidade, giro) e dará como resultado um vetor de saída (velocidade, giro).

Seja *de* e *ds* os vetores de entrada e saída, respectivamente. O módulo "Follow Wall" pode ser definido também como uma função FW, tal que $ds = FW(de)$

5.5.2 Controle

De forma similar a *evitar obstáculos*, o comportamento *seguir paredes* relaciona um conjunto de etiquetas à diversos intervalos de domínio de Θ. Estes intervalos variam em função do lado pelo qual se deseja alinhar o robô com a parede, como é mostrado na figura 5.5 [Fon96].

Uma vez definidas estas etiquetas, o controle deste comportamento fica baseado na comparação das áreas que ocupam os intervalos frente à esquerda, atrás à esquerda e direita, no caso em que deseje-se alinhar o robô com a parede pela sua esquerda e frente à direita, atrás à direita e esquerda, no caso em que deseje alinhar o robô com a parede

pela sua direita, então, as áreas dos intervalos etiquetados serão calculados como [Fon96]:

$$Área(frente.à.direita) = \sum_{i=0}^{90} Ar(i)$$

$$Área(esquerda) = \sum_{i=90}^{270} Ar(i)$$

$$Área(atrás.à.direita) = \sum_{i=270}^{0} Ar(i)$$

onde Ar está relacionado com a leitura do sensores.

Assim, se por exemplo o objetivo é que o robô siga a parede pela sua direita, então se comprovará se a área *frente à direita* é maior que a *atrás à direita*. Se este for o caso, o controle tratará que o robô gire à esquerda. Pode-se observar que neste caso a parte superior esquerda do robô encontra-se mais perto da parede, portanto, o robô tratará de afastar-se ligeiramente dela. Em caso contrário, o agente *seguir paredes* tratará que o robô gire à direita, ou seja, o aproximará da parede. Por exemplo, estas instruções poderiam ser escritas em pseudo código como [Fon96]:

```
SeguirParedes(Alinhar)
 {
   if (Alinhar == Direita)
            {
               if (Área(frente à direita) > Área(atrás à direita))
                     Giro = Esquerda;
               else
                     Giro = Direita;
            }
   if (Alinhar == Esquerda)
            {
               if (Área(frente à esquerda) > Área(atrás à esquerda))
                     Giro = Direita;
               else
                     Giro = Esquerda;
            }
 }
```

Poderia acontecer que na proximidade do robô não exista nenhuma parede, ou inclusive que a parede encontre-se do lado contrário ao qual desejamos segui-la. Assim, o agente *seguir paredes* avaliará a área que se encontra do lado contrário ao desejado (expresso como direita ou esquerda na figura 5.5). Se esta área for maior que a soma das áreas dos intervalos definidos como *frente à esquerda* e *atrás à esquerda*, ou *frente à direita* e *atrás à direita*, o agente tratará de procurar a parede em direção ao lado contrário ao desejado até encontrá-la. Procurar a parede consiste simplesmente em girar na direção contrária áquela com que se deseje alinhar o robô para segui-la. Portanto, com este comportamento é possível conseguir uma grande flexibilidade devido a que independentemente da orientação do robô em relação à parede que se deseje seguir, o agente orientará automaticamente o robô na direção adequada [Fon96].

5.6 Conclusão

Neste capítulo foram apresentados os aspectos básicos que devem ser considerados para projetar o sistema de controle de um robô móvel que lhe permita realizar tarefas básicas de navegação tais como: *evitar obstáculos* e *seguir paredes*. No próximo capítulo, estas tarefas serão implementadas usando um controlador nebuloso, e na seqüência o controlador nebuloso para seguir paredes será otimizado através de um algoritmo genético.

Capítulo 6

Desenvolvimento e otimização dos controladores nebulosos

6.1 Introdução

Uma das mais primeiras aplicações de um controlador nebuloso na navegação de veículos autônomos foi reportada por Sugeno e Murakami [SM85]. Nesta aplicação, o controlador de Takagi e Sugeno [TS83] é usado no estacionamento de um carro. As regras lingüísticas foram derivadas modelando-se as ações do homem no momento de estacionar um veículo.

Assim, desde 1985 um grande número de pesquisas têm sido dirigidas à aplicação do controle nebuloso em robótica móvel, por exemplo [Saf98]. Atualmente, no campo da navegação de robôs móveis autônomos, são populares os controladores nebulosos adaptativos distribuídos, as redes neurais e neuro-nebulosas, assim como os algoritmos de controle baseados na computação evolutiva.

Neste capítulo serão descritos os experimentos realizados, utilizando um simulador do robô móvel Khepera (figura 6.1). Este simulador está disponível na Internet (http://wwwi3s.unice.fr/ om/khep-sim.html). Para maior informação ver o apêndice A e [MAG95].

O ambiente das simulações é mostrado na figura 6.2. Observa-se que à esquerda encontra-se o ambiente do Khepera e à direita são mostradas as leituras dos sensores e os valores de voltagem aplicados em cada motor (da roda esquerda e direita).

Em relação às simulações, primeiramente, será apresentado o controlador nebuloso (ver apêndice B) projetado para realizar duas tarefas de navegação: *evitar obstáculos* ("Avoid") e *seguir paredes* ("Follow Wall"), e depois o controlador nebuloso desenvolvido

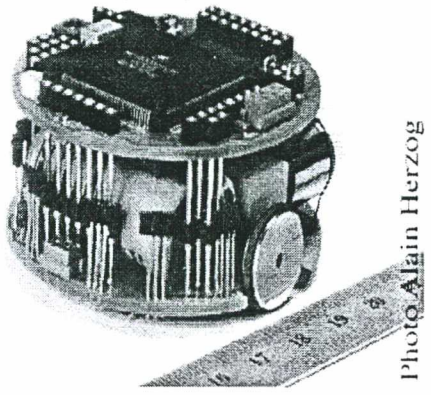


Figura 6.1: Robô Móvel Khepera.

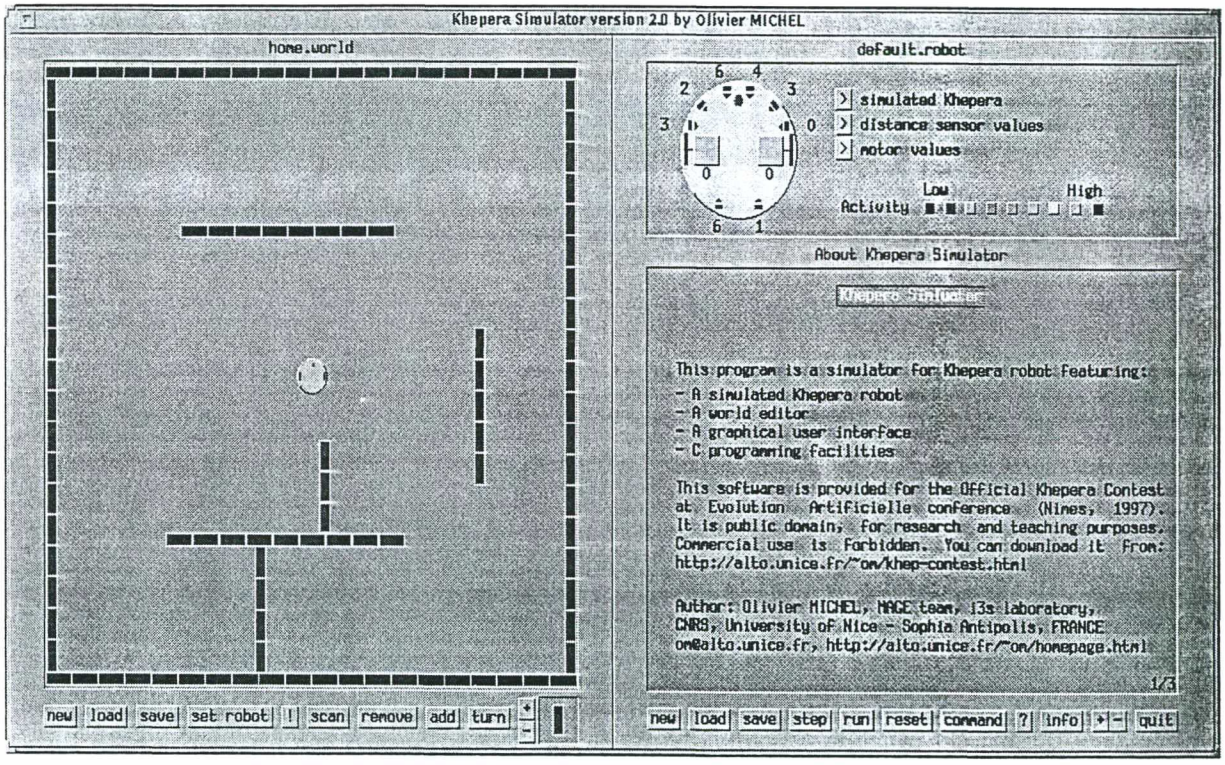


Figura 6.2: Ambiente das simulações.

para *seguir paredes* será otimizado usando-se um algoritmo genético. Na última seção são discutidos os resultados das simulações.

6.2 Controlador nebuloso para *evitar obstáculos*

Na maioria das tarefas de navegação o robô deve *evitar obstáculos*. Ele pode seguir um plano, transportar objetos, ou então, ir de um ponto a outro, mas em todas estas ações deve ser observada a segurança do robô e do ambiente.

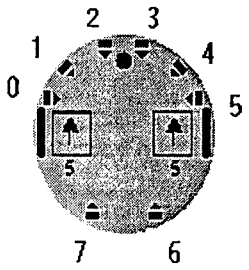


Figura 6.3: Representação do Khepera.

O objetivo desta seção é desenvolver um controlador nebuloso simples para *evitar obstáculos*. O robô Khepera [MAG95] tem 8 sensores infravermelhos e 2 motores. Portanto, o controlador deverá ter 8 entradas e duas saídas (ver figura 6.3). As entradas são as variáveis lingüísticas: d_{soi} =*distância entre o sensor e o obstáculo*, e as saídas são as variáveis lingüísticas: v_{mj} =*velocidade do motor*. Para cada variável lingüística podem ser definidos dois valores lingüísticos (ver figura 6.4):

- Livre (L)
- Colisão (C)

Cada saída pode ser definida com 3 valores lingüísticos:

- Atrás (As)
- Parar (Ps)
- Frente (Fs)

Estas definições significam que existem 256 (2^8) regras possíveis e, considerando as saídas, o total de combinações aumenta para 2304 ($2^8 * 3^2$). Evidentemente não é possível o desenvolvimento deste controlador sem uma metodologia precisa.

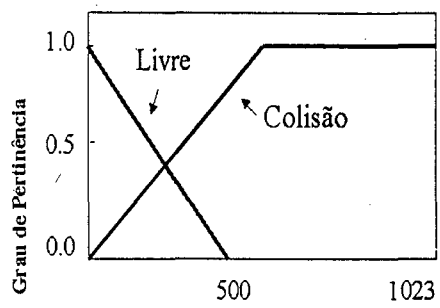


Figura 6.4: Funções de pertinência das entradas para *evitar obstáculos.*

Procurou-se, então, um menor número de funções de entrada que permitisse obter resultados adequados. Assim, foram definidas as seguintes entradas [God97]: *distância à frente (F), distância à esquerda (E), distância à direita (D)* e *distância atrás (A)*, sendo que estas distâncias são calculados usando as seguintes fórmulas :

$$\begin{array}{ll} \text{Frente} & F = \frac{S_2+S_3}{2} \\ \text{Esquerda} & E = \frac{S_0+S_1}{2} \\ \text{Direita} & D = \frac{S_4+S_5}{2} \\ \text{Atrás} & A = \frac{S_6+S_7}{2} \end{array}$$

onde S_0,S_1,S_2,S_3,S_4,S_5,S_6,S_7 representam as leituras dos sensores de proximidade do robô Khepera. As funções de pertinência correspondentes às entradas são mostradas na figura 6.4.

As leituras dos sensores variam no intervalo [0, 1023], onde 0 indica um afastamento de 5 cm e 1023 colisão total. Na prática os sensores de Khepera saturam-se a uma distância de 1 cm (900). Observe que agora o número de regras que corresponde às entradas é 16 (2^4).

A saída varia de -8 a +8, correspondentes à voltagem aplicada aos motores. Para as saídas, foi preciso estabelecer 5 funções de pertinência para obter melhores resultados (ver figura 6.5), onde NG é Negativo Grande, NP é Negativo Pequeno, Z é Zero, PP é Positivo Pequeno e PG é Positivo Grande (ver apêndice C).

As funções de pertinência associadas às variáveis lingüísticas de entrada são de tipo trapezoidal e de saída são de tipo triangular com exceção dos extremos que são de tipo trapezoidal. As regras lingüísticas foram definidas de maneira intuitiva e aprimoradas através de *tentativa e erro.* As regras foram baseadas em propriedades intuitivas dos movimentos, expressas por frases como:

- *Se* não existem obstáculos ao redor do robô, *então*, ir à frente;

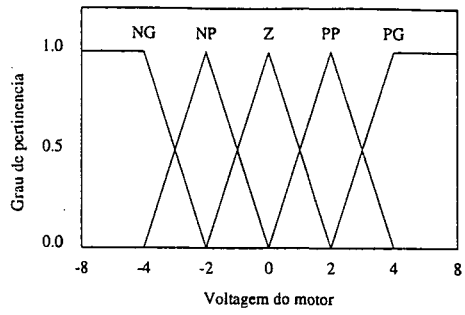


Figura 6.5: Funções de pertinência das saídas para *evitar obstáculos*.

Uma segunda:

- *Se* existe um obstáculo à esquerda do robô *e* não existe obstáculo à direita do robô, *então*, ir à direita;

E uma terceira:

- *Se* existe um obstáculo à direita do robô *e* não existe obstáculo à esquerda do robô, *então*, ir à esquerda.

Desta maneira foi desenvolvido a seguinte base de regras:

		MOTOR ESQUERDO	MOTOR DIREITO
1)	If FL and DL and EL and AL	then PG	then PG
2)	If FL and DL and EC and AL	then PG	then NG
3)	If FL and DC and EL and AL	then NG	then PG
4)	If FL and DC and EC and AL	then PP	then NP
5)	If FC and DL and EL and AL	then NP	then PP
6)	If FC and DL and EC and AL	then NP	then NP
7)	If FC and DC and EL and AL	then NP	then NP
8)	If FC and DC and EC and AL	then NG	then NG
9)	If FL and DL and EL and AC	then PG	then PG
10)	If FL and DL and EC and AC	then PG	then NG
11)	If FL and DC and EL and AC	then NG	then PG
12)	If FL and DC and EC and AC	then PP	then PP
13)	If FC and DL and EL and AC	then PP	then NP
14)	If FC and DL and EC and AC	then PP	then NP
15)	If FC and DC and EL and AC	then NP	then PP

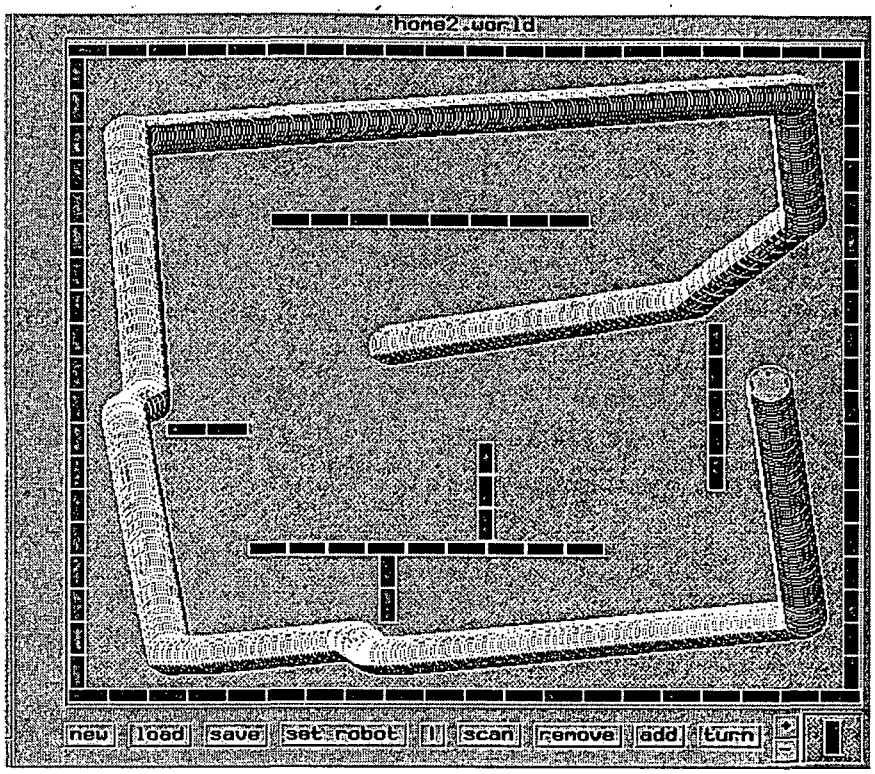


Figura 6.6: Khepera evitando obstáculos.

```
16)  If FC and DC and EC and AC        then  Z              then  Z
```

onde FL e FC representam Frente Livre e Frente Colisão; e assim por diante para as restantes variáveis lingüísticas: Esquerda, Direita e Atrás (ver figuras 6.4 e 6.5).

Esta base de regras, utilizada em um controlador nebuloso, permite ao robô movimentar-se em um ambiente simples sem ter um objetivo definido. As simulações são realizadas usando o método de Mandani ou Min-Max [Zad65] com desnebulização através do Centro de Gravidade.

Na figura 6.6 mostra-se o robô Khepera evitando obstáculos de forma satisfatória usando o controlador projetado. Deve ser observado que nem sempre um número superior de regras, por exemplo 81 (3^4), que corresponde a um número maior de funções de pertinência das entradas (3), resulta numa melhor solução [God97]. Destaca-se ainda, que quanto maior é o número de regras, maior é também o tempo de cálculo do controlador, e mais difícil a elaboração manual da base de regras de inferência.

Seguindo esta metodologia é possível projetar diferentes comportamentos mudando-se a base de regras. A próxima seção trata de outra tarefa que pode realizar um robô móvel: *seguir paredes*.

6.3 Controlador nebuloso para *seguir paredes*

Uma maneira simples de se tratar esta aplicação é considerar as seguintes variáveis lingüísticas: como entrada, a *distância* entre os sensores e o obstáculo; e como saída, a variável lingüística *velocidade de giro*. Por exemplo, no caso de *seguir paredes* pela direita a uma distância de 10 cm, podem ser estabelecidas as seguintes regras lingüísticas:

- *Se* a distância é menor que 10 cm, *então*, girar à esquerda;
- *Se* a distância é maior que 10 cm, *então*, o girar à direita ;
- Se a distância é aproximadamente 10 cm, *então*, não girar;

Assim, foram definidas três funções de pertinência para a variável lingüística *distância*: *Maior que 10 cm*, *Aproximadamente 10 cm* e *Menor que 10 cm*. Para o *ângulo de giro* também podem ser definidas três funções de pertinência, por exemplo: *Negativa* para girar à direita, *Positiva* para girar à esquerda e *Zero* para continuar sem girar.

Embora simples, este exemplo proporciona uma idéia geral das etapas do desenvolvimento de um controlador nebuloso para *seguir paredes*. A seguir é apresentado um controlador nebuloso mais completo para realizar esta tarefa de navegação. Novamente, considera-se que o robô vai seguir uma parede localizada à direita dele. Observe-se que a implementação do controlador para seguir uma parede pela esquerda é similar.

Para iniciar são definidas as seguintes variáveis lingüísticas (figura 6.7): W que representa a distância do robô à parede, G que representa a diferença entre a área à frente à direita e a área atrás à direita do robô. R que representa a diferença entre a área à esquerda e a área à direita do robô. Os valores lingüísticos de W, G e R são três: negativo (N) , zero (Z) e positivo (P). As funções de pertinência das saídas são definidas de forma similar às de evitar obstáculos (figura 6.5) (ver apêndice D).

Baseado nestas definições, a base de regras que foi desenvolvida para esta aplicação é a seguinte:

		MOTOR ESQUERDO	MOTOR DIREITO
1)	If RN and GZ and WZ	then PG	then PG
2)	If RN and GZ and WP	then PG	then NP
3)	If RN and GZ and WN	then NP	then PP
4)	If RN and GP and WZ	then NP	then PP
5)	If RN and GP and WP	then PP	then PP

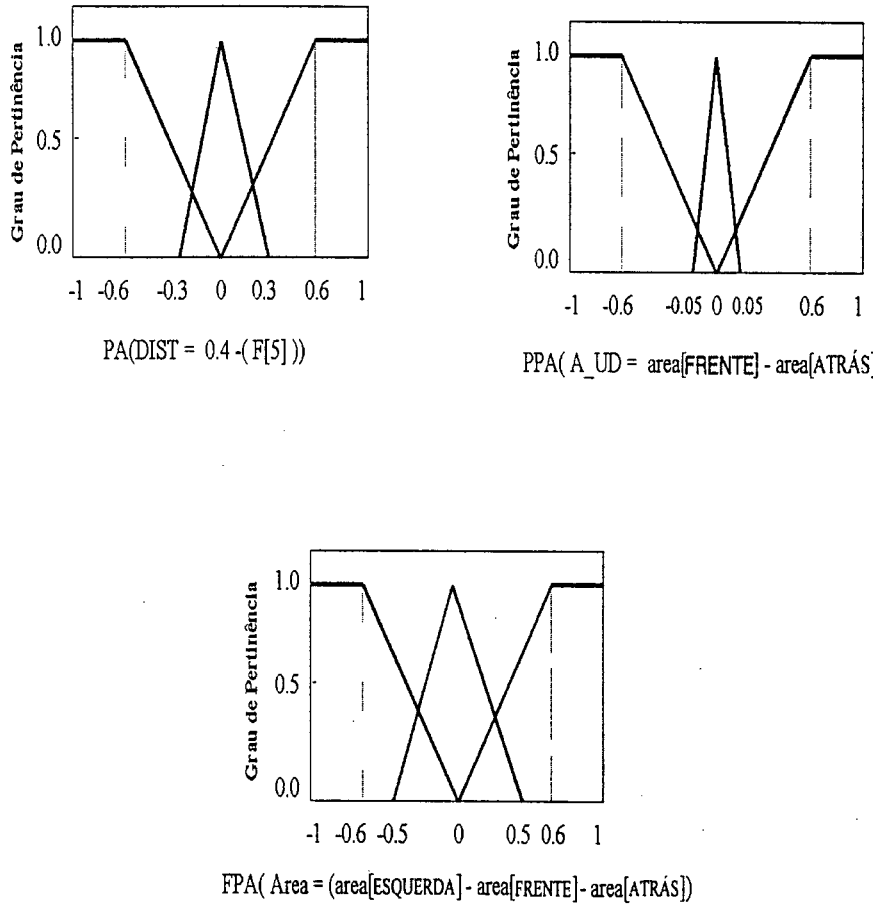


Figura 6.7: Funções de pertinência das entradas para *seguir paredes*.

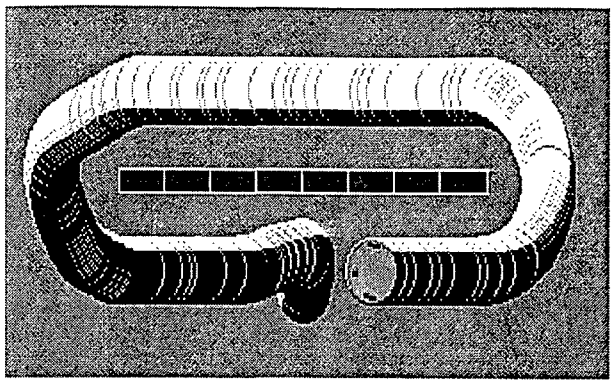


Figura 6.8: Khepera seguindo à parede pela direita.

6)	If RN and GP and WN	then NP	then PP
7)	If RN and GN and WZ	then PP	then NP
8)	If RN and GN and WP	then PP	then NP
9)	If RN and GN and WN	then PP	then PP
10)	If RP or RZ	then NP	then PP

onde WN, WZ e WP representam, respectivamente, W negativo, W zero e W positivo; e assim por diante para as restantes variáveis lingüísticas G e R.

Na figura 6.8, mostra-se o Khepera seguindo paredes pela direita. Destaca-se que tanto para *seguir paredes* quanto para *evitar obstáculos*, um problema crítico no projeto do controlador nebuloso foi a escolha das funções de pertinência e as regras lingüísticas. Portanto, é importante dispor de um método adequado que facilite a escolha de parâmetros e regras. A próxima seção propõe uma metodologia para otimizar a escolha das funções de pertinência do controlador nebuloso desenvolvido para *seguir paredes* através de um algoritmo genético.

6.4 Otimização das funções de pertinência do controlador nebuloso para *seguir paredes*

A idéia geral é otimizar os limites das funções de pertinência das saídas (voltagens aplicadas aos motores), usando-se um algoritmo genético. Para implementar este algoritmo foram usadas funções do toolbox SGA-C (ver apêndice E).

Ambos os softwares, o Simulador do robô Khepera, e o SGA-C foram modificados para comunicar-se através de arquivos, como se mostra na figura 6.9, além de terem sido adaptalos aos objetivos desta dissertação.

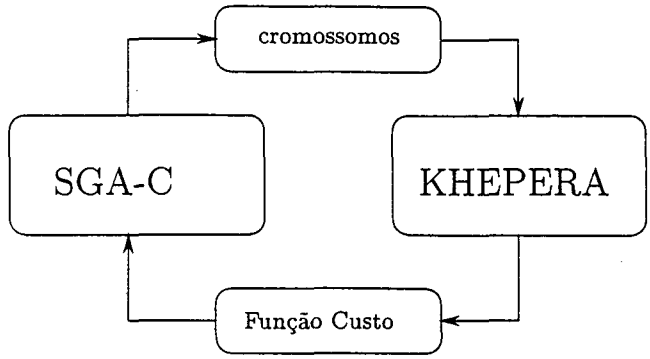


Figura 6.9: Comunicação entre Khepera e o Algoritmo Genético.

Cabe ressaltar que foi considerado que as funções de pertinência são simétricas, portanto, nas figuras a seguir somente é representado o lado positivo, que é *espelhado* para os valores negativos correspondentes. O número de iterações (número de passos do robô) é 1000.

A seguir serão apresentadas quatro metodologias.

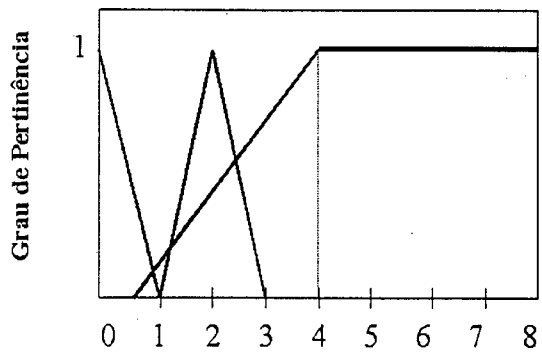
6.4.1 Primeira metodologia

A primeira metodologia tem por objetivo otimizar os limites inferior e superior para cada função de pertinência, mantendo os centros fixos.

A metodologia consiste nos seguintes passos:

1. Gerar uma população inicial de N cromossomos;
2. Avaliar a população, usando uma função custo;
3. Aplicar os operadores genéticos (crossover, mutação) com taxas P_c e P_m;
4. Aplicar o operador de seleção, baseado no método da roleta;
5. Gerar uma nova população e as respectivas funções de pertinência;
6. Repetir o experimento a partir do segundo passo até que o número de gerações seja igual a 50.

Os resultados desta simulação, com N=30, P_c=0.80, P_m=0.01, e a função custo definida como a somatória do erro da distância à parede ao quadrado, são mostrados na figura



b=1, a=1,c= 3.5, os centros fixos 0, 2 e 4

Figura 6.10: Funções de pertinência segundo a primeira metodologia.

6.10. Observa-se que existe um excesso de superposição das funções de pertinência, o que não é um resultado bom, para isto não acontecer poderia ser utilizada a "punição" dando valores de fitness "ruins" para os cromossomos com configurações "estranhas".

Observe-se que o algoritmo genético procura a melhor solução maximizando a função custo (minimizando o erro) usando a seguinte expressão:

$$F = \frac{1}{ERRO^2}$$

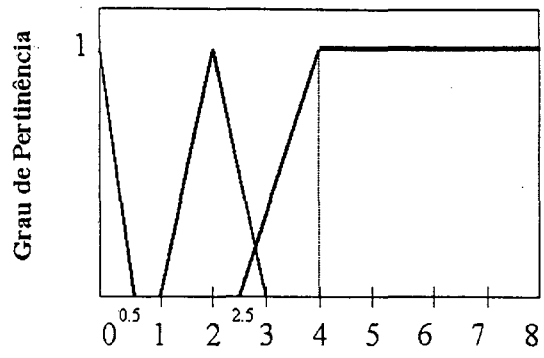
onde F é a função de avaliação e $ERRO^2$ é a somatória das diferenças entre a distância ideal até a parede e a distância determinada pela posição do robô a cada momento, ao quadrado.

Observou-se também que o comportamento do índice de aptidão tende a oscilar, o que implica que as populações deixam de alcançar um índice de aptidão crescente (ver figura 6.14). Para tentar resolver este problema foi proposta uma segunda metodologia.

6.4.2 Segunda metodologia

Os passos do algoritmo são similares à primeira metodologia, modificando apenas o objetivo da otimização. Neste caso, além de variar os limites inferior e superior de cada função de pertinência, também se varia a posição dos centros.

Os resultados desta simulação são mostrados na figura 6.11. Mesmo que resolvido o problema da superposição, aparece uma descontinuidade na saída, o que significa que entre 0.5 e 1 todos os conjuntos nebulosos têm grau de pertinência zero. E observou-se novamente um comportamento oscilatório no índice de aptidão (ver figura 6.14). Portanto, este resultado não é ainda um resultado bom. A seguir, apresenta-se uma terceira



b=1, a=0.5,c= 1.5, cpl=2, cph=4 e fixos centro 0

Figura 6.11: Funções de pertinência para a segunda metodologia.

metodologia que tenta superar as deficiências das duas anteriores.

6.4.3 Terceira metodologia

Agora as funções de pertinência são modificadas. Novamente os centros de cada função e sua amplitude variam, mas diferentemente do método anterior, são estabelecidas as seguintes relações:

$$x = 1 + \frac{g_1}{3} \qquad y = 4 + \frac{g_2}{3}$$

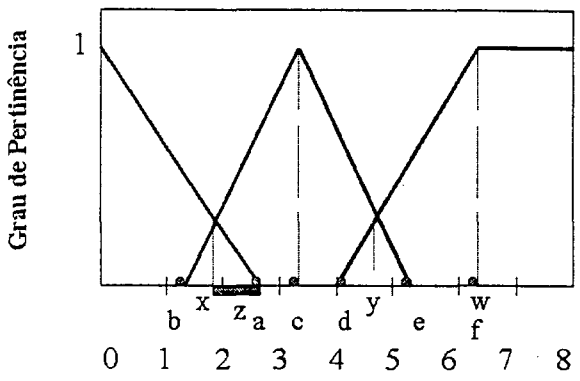
$$w = 6 + \frac{g_3}{3} \qquad z = \frac{g_4}{3}$$

onde g_1, g_2, g_3 e g_4 são valores determinados pelo algoritmo genético, que variam entre 0 e 3 (00 a 11 binário).

$$a = x + z \qquad b = x - z \qquad c = \frac{b+e}{2}$$

$$d = y - z \qquad e = y + z \qquad f = w$$

Observa-se que neste algoritmo foram superadas as limitações encontradas nas metodologias anteriores em relação aos resultados das funções de pertinência (figura 6.12), mas observou-se novamente que o comportamento do índice de aptidão não é crescente (figura 6.14).



b=1.33, a=2.67,c= 3.33, d=4, e=5.34, f=6.33

Figura 6.12: Funções de pertinência para a terceira metodologia.

6.4.4 Quarta metodologia

Agora realizaremos outra metodologia para que os conjuntos nebulosos mantenham uma forma de "sanfona" [Cor99]. O ponto de centro de um conjunto corresponde respectivamente ao ponto final da base do conjunto anterior e ao ponto inicial da base do próximo conjunto. Os parâmetros obtidos na aprendizagem com o AG são tratados como porcentagens do intervalo de busca, tomando-se como referência o centro definido em zero. Assim o primeiro conjunto terá como centro a própria porcentagem encontrada, o segundo será o centro anterior mais a porcentagem obtida com o AG e assim sucessivamente. O valor de a está no intervalo entre 1 a 4 e o valor de b está no intervalo entre 1,5 a 7,5.

$$a = 1 + \frac{g_1}{7} * 3$$

$$b = (a + 0.5) + \frac{g_2}{7} * 3$$

onde g_1 e g_2 são valores determinados pelo algoritmo genético, que variam entre 0 e 7 (000 a 111 binário).

Observa-se que neste algoritmo foram superadas as limitações encontradas nas dois primeiras metodologias em relação aos resultados das funções de pertinência (figura 6.13), mas observou-se novamente que o comportamento do índice de aptidão não é crescente (figura 6.14).

6.4.5 Comparação dos métodos de seleção

Para superar o problema do comportamento do índice de aptidão não crescente foram repetidos os passos da terceira metodologia, mas usando os outros dois métodos de se-

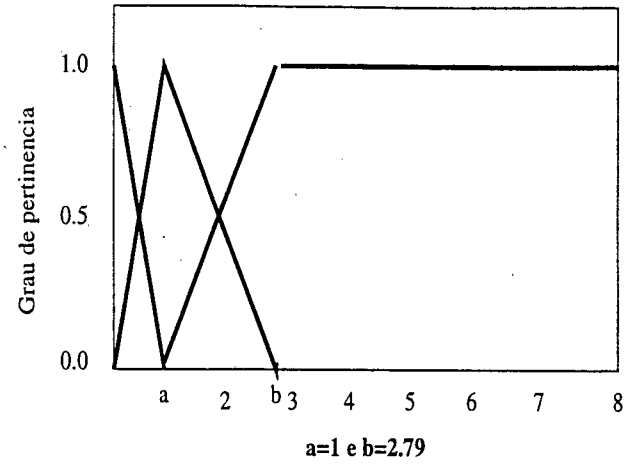


Figura 6.13: Funções de pertinência para a quarta metodologia.

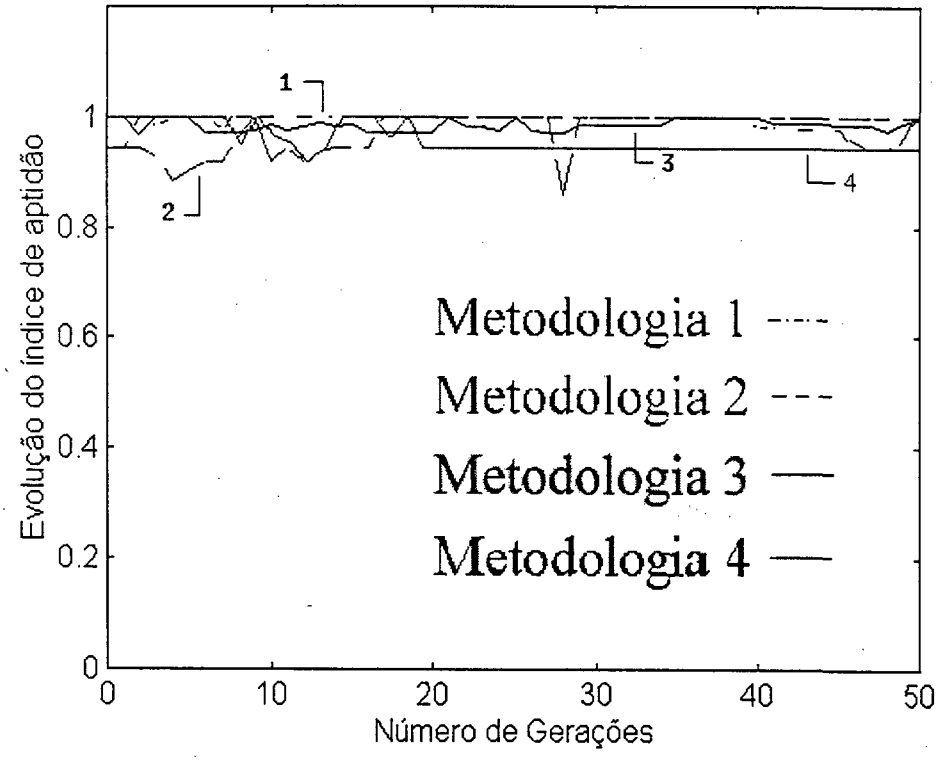


Figura 6.14: Evolução do máximo nas metodologias apresentadas.

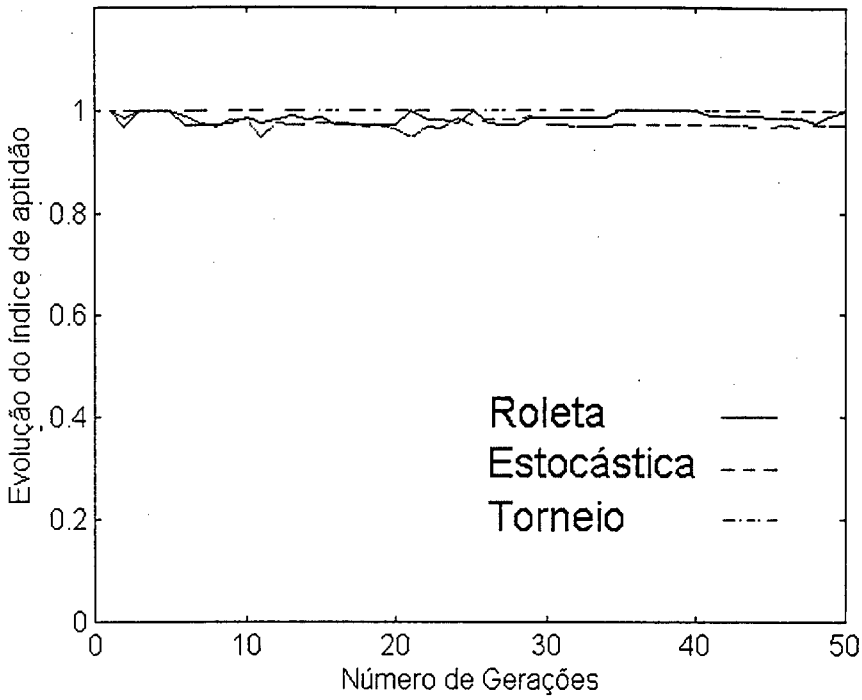


Figura 6.15: Evolução do máximo através da aplicação dos três métodos de seleção.

leção contidos no toolbox SGA-C: *torneio*, do inglês *tournament selection* [Bri81] e *sobra estocástica*, do inglês *stochastic-remainder selection* [Boo82].

Na figura 6.15 são mostrados os resultados da aplicação dos três métodos de seleção na terceira metodologia (foi incluído o resultado obtido através do método da *roleta*).

Somente o método de torneio [Bri81] não apresentou um comportamento oscilatório, sendo, portanto, o melhor resultado.

6.4.6 Comparação da terceira e quarta metodologia usando o método de seleção Torneio

Observa-se os resultados das funções de pertinência da terceira metodologia (figura 6.16), e quarta metodologia (figura 6.17), usando-se o método de seleção Torneio.

A evolução do máximo através da aplicação do método de seleção Torneio para a quarta metodologia não apresentou um comportamento oscilatório, do mesmo modo que a terceira metodologia que foi desenvolvida no subseção anterior.

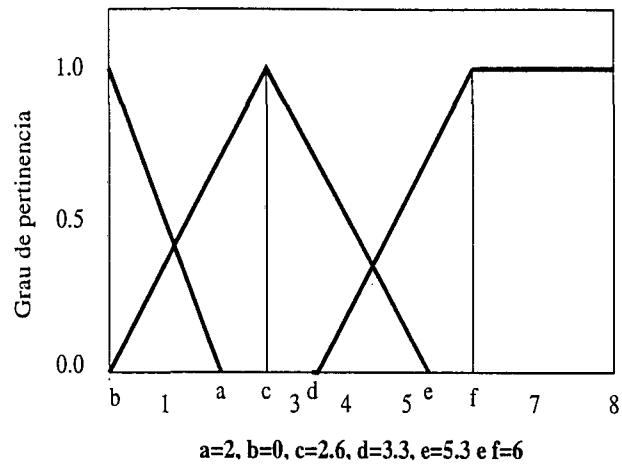


Figura 6.16: Funções de pertinência para a terceira metodologia.

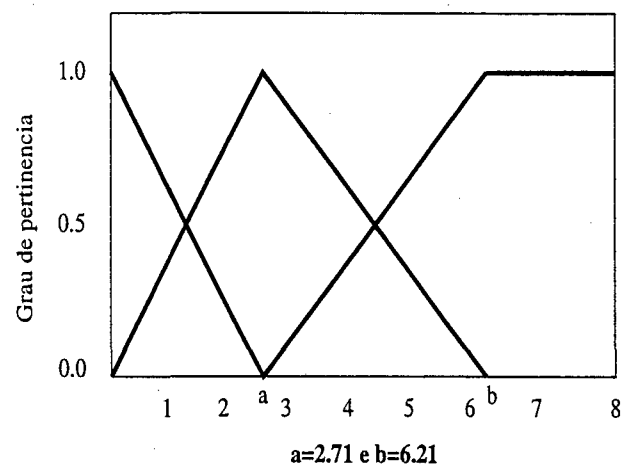


Figura 6.17: Funções de pertinência para a quarta metodologia.

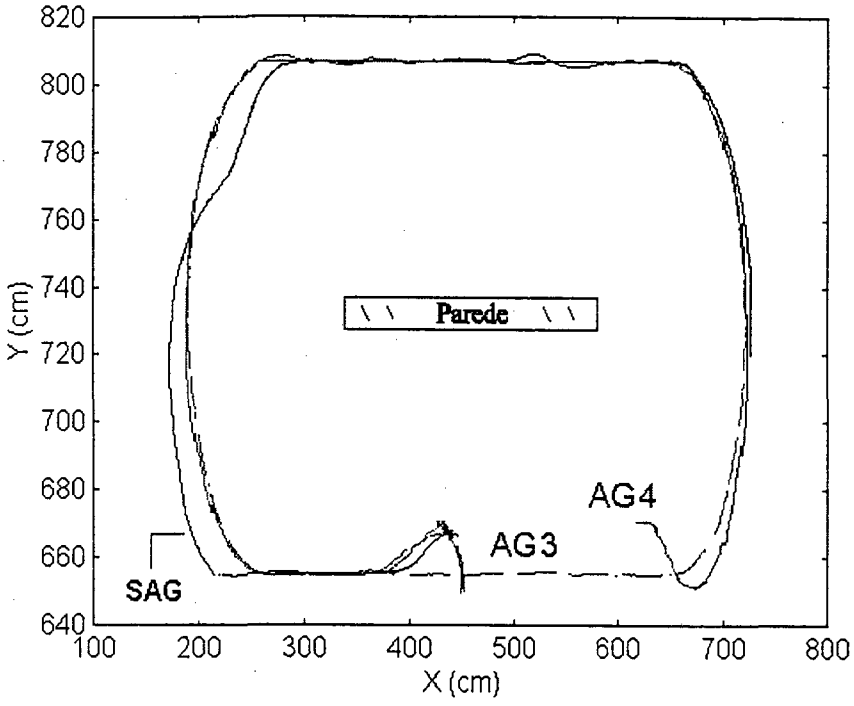


Figura 6.18: Controlador nebuloso feito a mano versus controladores nebulosos otimizados através do algoritmo genético.

6.4.7 Comparação das melhores soluções obtidas com o desenvolvido inicialmente

Para finalizar foram comparadas as melhores soluções obtidas através do método de torneio na terceira metodologia (**AG3**) e na quarta metodologia (**AG4**), com o controlador nebuloso desenvolvido inicialmente para *seguir paredes* (**SAG**). Observa-se na figura 6.18, que a resposta do controlador nebuloso otimizado através do algoritmo genético usando a terceira metodologia foi melhor, pois apresentou um menor erro acumulativo 109.58, para 129.23 do controlador nebuloso desenvolvido manualmente e para um 132.78 usando a quarta metodologia.

6.5 Conclusão

Neste capítulo foram tratados o projeto e simulação das metodologias apresentadas neste trabalho. Observa-se a aplicabilidade de ambos os controladores nebulosos, desenvolvidos para realizar tarefas de navegação em robótica móvel. Em particular, foram estudados os comportamentos *evitar obstáculos* e *seguir paredes*. A aplicação do algorit-

mo genético foi também um aspecto importante deste trabalho, e mostrou que através da otimização das funções de pertinência é possível chegar a um controlador mais robusto e eficiente.

Porém, o tempo de busca da melhor solução é a principal desvantagem desta metodologia. Se recomenda para trabalhos futuros realizar a otimização da base de regras [God97]. Recomenda-se também otimizar as funções de pertinência das entradas, usando a metodologia desenvolvida neste capítulo.

Capítulo 7

Conclusões gerais e perspectivas

7.1 Conclusões

Nesta dissertação foi estudado o projeto e implementação de duas metodologias da inteligência computacional. Em particular, foram tratadas a Lógica Fuzzy e a Computação Evolutiva. Estas técnicas foram aplicadas ao campo da robótica móvel para a realização de tarefas básicas de navegação. Foi utilizado um simulador do robô móvel Khepera.

O Capítulo 2 introduziu estas metodologias de controle com a definição de métodos para o tratamento de incerteza, assim como uma breve referência à Lógica Fuzzy e à Computação Evolutiva.

No Capítulo 3 foi mostrado como a Lógica Fuzzy proporciona uma ferramenta matemática elegante para tratar com fatos e regras, e com hipóteses e conclusões no contexto particular da incerteza. Esta abordagem é bem diferente da Lógica de Boole e sua formulação é mais complexa. Devido à falta de uma metodologia de desenvolvimento sistemático e a dificuldade do ajuste dos parâmetros do controlador fuzzy em sistemas complexos, foi aberto o caminho para tratar uma metodologia de otimização que permita superar esses problemas.

Assim, no Capítulo 4, inicia-se o estudo de uma das metodologias de otimização que mais têm atraído a atenção dos pesquisadores em Inteligência Artificial nos últimos anos, o Algoritmo Genético. Assim, neste trabalho foram otimizadas as funções de pertinência do controlador fuzzy proposto para realizar uma tarefa de navegação: *seguir paredes.*

No Capítulo 5 foram apresentados alguns aspectos teóricos e de implementação relacionados ao controle de Robôs Móveis Autônomos com o objetivo de aplicar os controladores estudados na realização de tarefas de navegação básicas, tais como: *seguir paredes* e *evitar obstáculos.* Este Capítulo é o ponto de união dos objetivos desta dissertação: o Controle de Robôs Móveis Autônomos, o Controle Fuzzy e os Algoritmos Genéticos.

No Capítulo 6 foram apresentados os resultados das simulações. Foi desenvolvido a aplicabilidade dos controladores nebulosos para realizar tarefas de navegação em robótica móvel. Em particular, foram estudados os comportamentos *evitar obstáculos* e *seguir paredes*. A aplicação do algoritmo genético foi também um aspecto importante deste trabalho, onde foi desenvolvido quatros metodologias até obter o melhor resultado e mostrando que através da otimização das funções de pertinência é possível chegar a um controlador mais robusto e eficiente.

7.2 Perspectivas para trabalhos futuros

A investigação de formas de projetos e metodologias da inteligência computacional no desenvolvimento de controladores apresenta inúmeras possibilidades. Ainda existem muitos aspectos que merecem estudos mais aprofundados dentro do contexto das temáticas tratadas nesta dissertação.

Destaca-se a área de agentes autônomos e robótica, onde existe um vasto campo de aplicações no espectro das metodologias da inteligência computacional. Por exemplo, seria interessante aprofundar a modelagem de outros comportamentos de navegação mais complexos através de *Sistemas Multiagentes* [CB98], [Saf98]; *Sistemas Neuro-fuzzy* [God97]; *Arquiteturas Híbridas* [Fon96]; e a *Solução Distribuída de Problemas* [Coe97].

Apêndice A

O robô móvel Khepera

A.1 Especificações do Khepera [MAG95]:

Caraterísticas: Pequeno / Modularidade / Elevada potência computational / Controle simples / Baixo custo.

Aplicações em Robôs Móveis: Experimentos numa área pequena (uma mesa) / Experimentos implicando vários robôs / Portátil (laboratórios, conferências, etc).

Experimentos: Evitar obstáculos / Seguir paredes / Ir de um ponto à outro / Outros comportamentos.

Tabela A.1: Especificações técnicas do Khepera.

ELEMENTOS	INFORMAÇÃO TÉCNICA
Processador	Motorola 68331
RAM	256 Kbytes
ROM	256 or 512 Kbytes
Motores	2 motores DC com encoder incremental (10 pulsos por mm de avanço do robô)
Sensores	8 sensores proximidade infra-vermelhos e sensores de luz
Energia	Baterias NiCd ou fonte externa
Autonomia	30 minutos (configuração com máxima atividade)
Bus	Pode ser expandido através de módulos adicionados ao "K-Extension bus".
Tamanho	Diâmetro: 55 mm Comprimento: 30 mm
Peso	70 g

Apêndice B

Documentação sobre Fuzzy.h e Fuzzy.c (RICE 4.0x)

```
RICE 4.0x  Copyright (C) 1993  Rene' Jager
Arquivo:   fuzzy.h
Autor: Rene' Jager
Data: November 16, 1992
```

B.1 Introdução

Este anexo resume brevemente as funções implementadas em FUZZY.H e FUZZY.C, os quais são arquivos que pertencem ao pacote de programas RICE [Del93].

B.2 Principais funções

Gerar funções de pertinência

```
float fpi(float val, float p1, float p2, float p3, float p4);

#define fdelta(val, p1, p2, p3)    fpi(val, p1, p2, p2, p3)
#define fgamma(val, p1, p2)        fpi(val, p1, p2, p2, p2)
#define flambda(val, p1, p2)       fpi(val, p1, p1, p1, p2)
```

```
float ftrapezium(float val, float p1, float p2, float p3, float p4);

#define fblock(val, p1, p2)          ftrapezium(val, p1, p1, p2, p2)
#define ftriangle(val, p1, p2, p3)   ftrapezium(val, p1, p2, p2, p3)
```

Métodos de desnebulização

```
float ficog(int len, float *set, float lim);
float fcog(int len, float *set);
float fmom(int len, float *set);
```

Operador intersecção

```
float *fzand(int len, float *dest, float *src);
float *fland(int len, float *dest, float *src);
float *fpand(int len, float *dest, float *src);
```

Operador união

```
float *fzor(int len, float *dest, float *src);
float *flor(int len, float *dest, float *src);
float *fpor(int len, float *dest, float *src);
```

Operador negação

```
float *fnot(int len, float *dest);
```

Altura

```
float fhgt(int len, float *src);
```

Preenche matriz com trapezoide discretizado

```
float *fcriatrapezoide(int len, float *dest, float start, float end, float p1, f
```

Limita valor máximo de valores dentro de uma matriz

```
float *flim(float len, float *dest, float hmax);
```

Apêndice C

Algoritmo para Evitar Obstáculos ("Avoid")

```
/*****************************************************************/
/* File:         user.c (Khepera Simulator)                      */
/* Author:       Amarilys Lima Lopez <amarilys@lcmi.ufsc.br>     */
/* Date:         Thu Outubre 30 14:39:05 1998                    */
/* Description: example of user.c file EVITAR OBSTACULOS         */
/*****************************************************************/

#include "../SRC/include.h"
#include "user_info.h"
#include "user.h"
#include "fuzzy.c"

/* Definições Iniciais */

#define ESQUERDA 0
#define FRENTE   1
#define DIREITA  2
#define ATRAS    3

/* Funções de Pertinência das Entradas */

#define LIVRE_P1 0
```

```
#define LIVRE_P2 0
#define LIVRE_P3 0
#define LIVRE_P4 500

#define COLISAO_P1 0
#define COLISAO_P2 500
#define COLISAO_P3 1023
#define COLISAO_P4 1023

/* Funções de Pertinência das Saídas */

#define M_FRENTE_L_P1 0
#define M_FRENTE_L_P2 2
#define M_FRENTE_L_P3 2
#define M_FRENTE_L_P4 4

#define M_FRENTE_H_P1 2
#define M_FRENTE_H_P2 4
#define M_FRENTE_H_P3 6
#define M_FRENTE_H_P4 6

#define M_TRAS_L_P1 0
#define M_TRAS_L_P2 -2
#define M_TRAS_L_P3 -2
#define M_TRAS_L_P4 -4

#define M_TRAS_H_P1 -2
#define M_TRAS_H_P2 -4
#define M_TRAS_H_P3 -6
#define M_TRAS_H_P4 -6

#define M_STOP_P1 -2
#define M_STOP_P2 0
#define M_STOP_P3 0
#define M_STOP_P4 2
```

```
#define INICIO_INTERV_SAIDA -6
#define FIM_INTERV_SAIDA     6
#define PASSO_DISCRET_SAIDA .1

#define TAM_MATRIZ_SAIDA    121
#define VECES 9000

int cont=0;

float *saida;
float *aux;
FILE *path_file;

void UserInit(struct Robot *robot)
{
saida = malloc(sizeof(float[TAM_MATRIZ_SAIDA]));
aux = malloc(sizeof(float[TAM_MATRIZ_SAIDA]));
}

void UserClose(struct Robot *robot)
{
free(saida);
free(aux);
}

void RunRobotStart(struct Robot *robot)
{
  path_file = fopen("STATS/path_1.m","a");
fprintf(path_file,"A=[");
}

void RunRobotStop(struct Robot *robot)
{
        fprintf(path_file,"];");
fclose(path_file);
cont = 0;
```

```
}

boolean StepRobot(struct Robot *robot)
{
int sensor[4]={0,0,0,0};
float EL,EC,FL,FC,DL,DC,AL,AC;
float regra[16];
int k;

/* Toma como valor dos sinais de entrada a media dos valores dos
sensores */

sensor[ESQUERDA]=
(robot->IRSensor[0].DistanceValue+robot->IRSensor[1].DistanceValue)/2;
sensor[FRENTE]=
(robot->IRSensor[2].DistanceValue+robot->IRSensor[3].DistanceValue)/2;
sensor[DIREITA]=
(robot->IRSensor[4].DistanceValue+robot->IRSensor[5].DistanceValue)/2;
  sensor[ATRAS]=
(robot->IRSensor[6].DistanceValue+robot->IRSensor[7].DistanceValue)/2;


/* Calcula grau de pertinencia das variaveis de entrada */

EL=
ftrapezium(sensor[ESQUERDA],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
EC=
ftrapezium(sensor[ESQUERDA],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
FL=
ftrapezium(sensor[FRENTE],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
FC=
ftrapezium(sensor[FRENTE],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
DL=
ftrapezium(sensor[DIREITA],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
DC=
ftrapezium(sensor[DIREITA],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
```

```
        AL=
ftrapezium(sensor[ATRAS],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
AC=
ftrapezium(sensor[ATRAS],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);

/* Aplica Regras */

regra[0] =MIN(MIN(MIN(FL, DL), EL), AL);
regra[1] =MIN(MIN(MIN(FL, DL), EC), AL);
regra[2] =MIN(MIN(MIN(FL, DC), EL), AL);
regra[3] =MIN(MIN(MIN(FL, DC), EC), AL);
regra[4] =MIN(MIN(MIN(FC, DL), EL), AL);
regra[5] =MIN(MIN(MIN(FC, DL), EC), AL);
regra[6] =MIN(MIN(MIN(FC, DC), EL), AL);
regra[7] =MIN(MIN(MIN(FC, DC), EC), AL);
regra[8] =MIN(MIN(MIN(FL, DL), EL), AC);
regra[9] =MIN(MIN(MIN(FL, DL), EC), AC);
regra[10]=MIN(MIN(MIN(FL, DC), EL), AC);
regra[11]=MIN(MIN(MIN(FL, DC), EC), AC);
regra[12]=MIN(MIN(MIN(FC, DL), EL), AC);
regra[13]=MIN(MIN(MIN(FC, DL), EC), AC);
regra[14]=MIN(MIN(MIN(FC, DC), EL), AC);
regra[15]=MIN(MIN(MIN(FC, DC), EC), AC);

/* Calcula saida para o motor esquerdo */

/* Parcela relativa a ir para a frente H */
saida=
fcriatrapezoide(TAM_MATRIZ_SAIDA, saida,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_H_P1,M_FRENTE_H_P2,
M_FRENTE_H_P3,M_FRENTE_H_P4);
saida=
flim(TAM_MATRIZ_SAIDA, saida, MAX(MAX(MAX(regra[0] ,regra[1]),
regra[8]), regra[9]));

/* Parcela relativa a ir para a frente L */
```

```
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_L_P1,M_FRENTE_L_P2,
M_FRENTE_L_P3,M_FRENTE_L_P4);
aux=flim(TAM_MATRIZ_SAIDA, aux, MAX(regra[11], regra[13]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras H */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_H_P1,M_TRAS_H_P2,
M_TRAS_H_P3,M_TRAS_H_P4);
aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(MAX(MAX(MAX(regra[2], regra[5]),
regra[6]), regra[10]), regra[4]), regra[7]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para a tras L */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_L_P1,M_TRAS_L_P2,
M_TRAS_L_P3,M_TRAS_L_P4);
aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(regra[14], regra[12]), regra[3]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a parar o robot */
  aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_STOP_P1,M_STOP_P2,
M_STOP_P3,M_STOP_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, regra[15]);

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Calcula o centroide do conjunto de saida */
```

```
robot->Motor[LEFT].Value=
fcog(TAM_MATRIZ_SAIDA, saida)*PASSO_DISCRET_SAIDA+INICIO_INTERV_SAIDA;


/* Calcula saida para o motor direito */

/* Parcela relativa a ir para a frente H */
saida=fcriatrapezoide(TAM_MATRIZ_SAIDA, saida,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_H_P1,M_FRENTE_H_P2,
M_FRENTE_H_P3,M_FRENTE_H_P4);

saida=
flim(TAM_MATRIZ_SAIDA, saida, MAX(MAX(MAX(MAX(regra[0],regra[2]),
regra[8]),regra[10]), regra[4]));

/* Parcela relativa a ir para a frente L */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux, I
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_L_P1,M_FRENTE_L_P2,
M_FRENTE_L_P3,M_FRENTE_L_P4);

aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(regra[11], regra[14]), regra[12]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras H */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_H_P1,M_TRAS_H_P2,
M_TRAS_H_P3,M_TRAS_H_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(MAX(MAX(regra[1], regra[6]),
regra[9]), regra[5]), regra[7]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras L */
```

```
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_L_P1,M_TRAS_L_P2,M_TRAS_L_P3,
M_TRAS_L_P4);
aux=flim(TAM_MATRIZ_SAIDA, aux, MAX(regra[3], regra[13]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a parar o robot */
  aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_STOP_P1,M_STOP_P2,M_STOP_P3,
M_STOP_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, regra[15]);

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Calcula o centroide do conjunto de saida */
robot->Motor[RIGHT].Value=
fcog(TAM_MATRIZ_SAIDA, saida)*PASSO_DISCRET_SAIDA+INICIO_INTERV_SAIDA;

/* Armazena dados */

  fprintf(path_file,"%lg, %lg\n",robot->X,1000.0-robot->Y);

/* Determina se parar ou continuar */

cont++;
if (cont >= VECES)
  return(FALSE);
else
  return(TRUE);
}

void ResetRobot(struct Robot *robot)
{
 FILE *test;
```

```
  test = fopen("STATS/path_n.m","r");
  if (test)
  {
    fclose(test);
    system("rm STATS/path_n.m");
  }

}
```

Apêndice D

Algoritmo para Seguir Paredes ("Follow Wall")

```
/*****************************************************************/
/* File:         user.c (Khepera Simulator)                      */
/* Author:       Amarilys Lima Lopez<amarilys@lcmi.ufsc.br       */
/* Date:         Quarta Feira, Out 10:45:05 1998                 */
/* Description: example of user.c file SEGUIR PAREDES            */
/*****************************************************************/

#include "../SRC/include.h"
#include "user_info.h"
#include "user.h"
#include "fuzzy.c"

/* Definições Iniciais */

#define ESQUERDA 0
#define FRENTE   1
#define DIREITA  2
#define ATRAS    3

#define ACIMA    0
#define DEBAIXO  1
#define RESTO    2
```

```
/* Funções de Pertinência das Entradas */

#define LIVRE_P1 0
#define LIVRE_P2 0
#define LIVRE_P3 0
#define LIVRE_P4 500

#define COLISAO_P1 0
#define COLISAO_P2 500
#define COLISAO_P3 1023
#define COLISAO_P4 1023

#define ZERO_FPA -0.5
#define ZERO_FPB 0
#define ZERO_FPC 0
#define ZERO_FPD 0.5

#define MAIOR_FPA 0
#define MAIOR_FPB 0.6
#define MAIOR_FPC 2.0
#define MAIOR_FPD 2.0

#define MENOR_FPA 0
#define MENOR_FPB -0.6
#define MENOR_FPC -2.0
#define MENOR_FPD -2.0

#define ZERO_PPA -0.05
#define ZERO_PPB 0
#define ZERO_PPC 0
#define ZERO_PPD 0.05

#define MAIOR_PPA 0
#define MAIOR_PPB 0.6
#define MAIOR_PPC 1
```

```
#define MAIOR_PPD 1

#define MENOR_PPA 0
#define MENOR_PPB -0.6
#define MENOR_PPC -1
#define MENOR_PPD -1

#define ZERO_PA -0.3
#define ZERO_PB 0
#define ZERO_PC 0
#define ZERO_PD 0.3

#define MAIOR_PA 0
#define MAIOR_PB 0.6
#define MAIOR_PC 1
#define MAIOR_PD 1

#define MENOR_PA 0
#define MENOR_PB -0.6
#define MENOR_PC -1
#define MENOR_PD -1

/* Funções de Pertinência das Saídas */

#define M_FRENTE_L_P1 1
#define M_FRENTE_L_P2 2.8
#define M_FRENTE_L_P3 2.8
#define M_FRENTE_L_P4 4.5

#define M_FRENTE_H_P1 4
#define M_FRENTE_H_P2 5
#define M_FRENTE_H_P3 6
#define M_FRENTE_H_P4 6

#define M_TRAS_L_P1 -1
#define M_TRAS_L_P2 -2.8
```

```
#define M_TRAS_L_P3 -2.8
#define M_TRAS_L_P4 -4.5

#define M_TRAS_H_P1 -4
#define M_TRAS_H_P2 -5
#define M_TRAS_H_P3 -6
#define M_TRAS_H_P4 -6

#define M_STOP_P1 -2
#define M_STOP_P2 0
#define M_STOP_P3 0
#define M_STOP_P4 2

#define INICIO_INTERV_SAIDA -6
#define FIM_INTERV_SAIDA     6
#define PASSO_DISCRET_SAIDA .1

/* O tamanho da matriz de saida deve ser calculado pela expressao:
   (FIM_INTERV_SAIDA-INICIO_INTERV_SAIDA)/PASSO_DISCRET_SAIDA+1

#define TAM_MATRIZ_SAIDA    121
#define VECES 4000
int cont=0;
int i;

float *saida;
float *aux;
float DIST=0;
float AREA=0;
float A_UD=0;
float EL,EC,FL,FC,DL,DC,AL,AC;
float PP,PZ,PN, PPA, PZA, PNA;
float FZE,FMA,FME;
float regras[11];
int j;
int N;
```

```
FILE *path_file;

void UserInit(struct Robot *robot)
{
saida = malloc(sizeof(float[TAM_MATRIZ_SAIDA]));
aux = malloc(sizeof(float[TAM_MATRIZ_SAIDA]));
}

void UserClose(struct Robot *robot)
{
free(saida);
free(aux);
}

void RunRobotStart(struct Robot *robot)
{
path_file = fopen("STATS/path_n.m","a");
fprintf(path_file,"A=[");
}

void RunRobotStop(struct Robot *robot)
{
        fprintf(path_file,"];");
fclose(path_file);
        cont = 0;
}

boolean StepRobot(struct Robot *robot)
{
int sensor[4]={0,0,0,0};
float area[3]={0,0,0};
float F[8];

/* Toma como valor dos sinais de entrada a media dos valores dos
sensores */
```

```
sensor[ESQUERDA]=
(robot->IRSensor[0].DistanceValue+robot->IRSensor[1].DistanceValue)/2;

sensor[FRENTE]=
(robot->IRSensor[2].DistanceValue+robot->IRSensor[3].DistanceValue)/2;

sensor[DIREITA]=
(robot->IRSensor[4].DistanceValue+robot->IRSensor[5].DistanceValue)/2;

sensor[ATRAS]=
(robot->IRSensor[6].DistanceValue+robot->IRSensor[7].DistanceValue)/2;


/* Etiquetas linguisticas Arriba, abajo y resto */

 for (i=0;i<8;i++)
       {
 /* normaliza leituras */
         F[i]= (float)robot->IRSensor[i].DistanceValue / 1023.0;
       }

area[ACIMA]=F[3];
area[DEBAIXO]=F[6];
area[RESTO]=((F[0]+F[1])/2 + (F[2]+F[7])/2);

/* Calcula grau de pertinencia das variaveis de entrada */
EL=
ftrapezium(sensor[ESQUERDA],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
EC=
ftrapezium(sensor[ESQUERDA],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
FL=
ftrapezium(sensor[FRENTE],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
FC=
ftrapezium(sensor[FRENTE],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
DL=
```

```
ftrapezium(sensor[DIREITA],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
DC=
ftrapezium(sensor[DIREITA],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);
        AL=
ftrapezium(sensor[ATRAS],LIVRE_P1,LIVRE_P2,LIVRE_P3,LIVRE_P4);
AC=
ftrapezium(sensor[ATRAS],COLISAO_P1,COLISAO_P2,COLISAO_P3,COLISAO_P4);

/* Calcula grau de pertinencia das novas variaveis de entrada */

DIST = 0.4 - (F[5]);
AREA = (area[RESTO] -  area[ACIMA] - area[DEBAIXO]);
A_UD = area[ACIMA]- area[DEBAIXO] ;

PPA=ftrapezium(A_UD,MAIOR_PPA,MAIOR_PPB,MAIOR_PPC,MAIOR_PPD);
        PZA=ftrapezium(A_UD,ZERO_PPA,ZERO_PPB,ZERO_PPC,ZERO_PPD);
        PNA=ftrapezium(A_UD,MENOR_PPA,MENOR_PPB,MENOR_PPC,MENOR_PPD);

  PP=ftrapezium(DIST,MAIOR_PA,MAIOR_PB,MAIOR_PC,MAIOR_PD);
        PZ=ftrapezium(DIST,ZERO_PA,ZERO_PB,ZERO_PC,ZERO_PD);
        PN=ftrapezium(DIST,MENOR_PA,MENOR_PB,MENOR_PC,MENOR_PD);

FZE=ftrapezium(AREA,ZERO_FPA,ZERO_FPB,ZERO_FPC,ZERO_FPD);
FMA=ftrapezium(AREA,MAIOR_FPA,MAIOR_FPB,MAIOR_FPC,MAIOR_FPD);
FME=ftrapezium(AREA,MENOR_FPA,MENOR_FPB,MENOR_FPC,MENOR_FPD);


/* Aplica regras para seguir paredes pela direita */

        regras[0] =MIN(MIN(PZ, PZA), MAX(FZE, FME));
regras[1] =MIN(MIN(PP, PZA), MAX(FZE, FME));
        regras[2] =MIN(MIN(PN, PZA), MAX(FZE, FME));
regras[3] =MIN(MIN(PZ, PPA), MAX(FZE, FME));
regras[4] =MIN(MIN(PN, PPA), MAX(FZE, FME));
regras[5] =MIN(MIN(PP, PPA), MAX(FZE, FME));
regras[6] =MIN(MIN(PZ, PNA), MAX(FZE, FME));
```

```
regras[7] =MIN(MIN(PN, PNA), MAX(FZE, FME));
regras[8] =MIN(MIN(PP, PNA), MAX(FZE, FME));
        regras[9] =MIN(AL, FMA);
        regras[10]=MIN(MIN(MIN(FC, DC), EC), AC);/*rodeado de obtáculos*/

/* Calcula saida para o motor esquerdo */

/* Parcela relativa a ir para a frente H */
saida=
fcriatrapezoide(TAM_MATRIZ_SAIDA, saida,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_H_P1,M_FRENTE_H_P2,
M_FRENTE_H_P3, M_FRENTE_H_P4);
        saida=flim(TAM_MATRIZ_SAIDA, saida, regras[0]);

/* Parcela relativa a ir para a frente L */
aux=
fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_L_P1,M_FRENTE_L_P2,
M_FRENTE_L_P3, M_FRENTE_L_P4);

aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(MAX(regras[1],regras[6]),regras[7]),
regras[8]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras H */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_H_P1,M_TRAS_H_P2,M_TRAS_H_P3,
M_TRAS_H_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, regras[9]);

  /*MAX(regras[9], regras[10]));*/

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);
```

```
/* Parcela relativa a ir para a tras L */
aux=
fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_L_P1,M_TRAS_L_P2,M_TRAS_L_P3,
M_TRAS_L_P4);

aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(MAX(regras[2],regras[3]),regras[4]),
regras[5]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a parar o robot */
aux=
fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_STOP_P1,M_STOP_P2,M_STOP_P3,
M_STOP_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, regras[10]);

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Calcula o centroide do conjunto de saida */

robot->Motor[LEFT].Value=
fcog(TAM_MATRIZ_SAIDA,saida)*PASSO_DISCRET_SAIDA+INICIO_INTERV_SAIDA;


/* Calcula saida para o motor direito */

/* Parcela relativa a ir para a frente H */
      saida=
fcriatrapezoide(TAM_MATRIZ_SAIDA, saida,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_H_P1,M_FRENTE_H_P2,
M_FRENTE_H_P3, M_FRENTE_H_P4);
```

```
saida=flim(TAM_MATRIZ_SAIDA, saida, MAX(regras[0], regras[9]));
        /* MAX(MAX(regras[0], regras[9]), regras[10]));*/

/* Parcela relativa a ir para a frente L */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,

INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_FRENTE_L_P1,M_FRENTE_L_P2,
M_FRENTE_L_P3, M_FRENTE_L_P4);

aux=
flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(MAX(MAX(regras[2],regras[3]),
regras[4]), regras[5]), regras[7]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras H */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_H_P1,M_TRAS_H_P2,M_TRAS_H_P3,
M_TRAS_H_P4);

aux=flim(TAM_MATRIZ_SAIDA, aux, 0.0);

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a ir para tras L */
aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_TRAS_L_P1,M_TRAS_L_P2,M_TRAS_L_P3,
M_TRAS_L_P4);
aux=flim(TAM_MATRIZ_SAIDA, aux, MAX(MAX(regras[1],regras[6]),
regras[8]));

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Parcela relativa a parar o robot */
  aux=fcriatrapezoide(TAM_MATRIZ_SAIDA, aux,
```

```
INICIO_INTERV_SAIDA,FIM_INTERV_SAIDA,M_STOP_P1,M_STOP_P2,M_STOP_P3,
M_STOP_P4);
aux=flim(TAM_MATRIZ_SAIDA, aux, regras[10]);

saida=fzor(TAM_MATRIZ_SAIDA, saida, aux);

/* Calcula o centroide do conjunto de saida */

robot->Motor[RIGHT].Value=
fcog(TAM_MATRIZ_SAIDA, saida)*PASSO_DISCRET_SAIDA+INICIO_INTERV_SAIDA;

/ * Armazena dados */

fprintf(path_file,"%f, %f, %f, %lg, %lg, %d, %d\n",

DIST, A_UD, AREA, robot->X,1000.0-robot->Y, robot->Motor[LEFT].Value,
robot->Motor[RIGHT].Value);

cont++;
if (cont >= VECES)
  return(FALSE);
else
  return(TRUE);
}

void ResetRobot(struct Robot *robot)
{
  FILE *test;
  test = fopen("STATS/path_n.m","r");
  if (test)
  {
    fclose(test);
    system("rm STATS/path_n.m");
  }
}
```

Apêndice E

SGA-C: Implementação em C de um Algoritmo Genético simples

Robert E. Smith Universidade de Alabama Departamento de Engenharia Mecânica
David E. Goldberg Universidade de Illinois Departamento de Engenheria
Jeff A. Earickson Rede de Supercomputadores de Alabama

E.1 Introdução

SGA-C é uma tradução e extensão do código original do SGA em Pascal [Cle94], desenvolvido por David Goldberg [Gol89].

E.2 Arquivos do SGA-C

sga.h inclui as declarações das estruturas e variáveis globais.

external.h declarações a serem incluídas em todos os códigos fontes, sem incluir **main()**.

main.c programa principal **main()**.

generate.c inclui **generation()**, um programa que gera e avalia as populações.

initial.c subrotinas que são chamadas no início de cada iteração do AG.

memory.c rotinas para a manipulação da memória.

operators.c inclui os operadores genéticos (cruzamento e mutação).

random.c utilitários para a manipulação de números aleatórios.

report.c imprime um relatório em cada ciclo de operação do AG.

rselect.c rotina para a seleção através do método da roleta.

srselect.c seleção estocástica [Boo82].

tselect.c seleção por torneio [Bri81].[1]

E.2.1 Entrada-Saída

O SGA-C permite execuções múltiplas do AG. Quando o programa é iniciado, o usuário deve entrar os seguintes dados:

- Número de corridas do AG (**int**).
- Tamanho da população (**int**).
- Número de cromossomos (**int**).
- Serão impressos os cromossomos em cada geração (**y/n**)?
- Número máximo de gerações em cada execução do AG (**int**).
- Probabilidade of cruzamento (**float**).
- Probabilidade of mutação (**float**).
- Alguma entrada relacionada com a aplicação específica (se houver).
- Valor entre 0 e 1, que será usado pela rotina de geração de números aleatórios (**float**).

E.3 Subrotinas Específicas da Aplicação

Para implementar uma aplicação do AG deverá ser criado um novo arquivo **app.c**. Se são necessárias novas variáveis para o problema específico, o método mais simples para que elas fiquem acessíveis aos outros módulos do SGA-C é declará-las em **sga.h** e **external.h**, tomando cuidado de não definí-las mais de uma vez.

[1] O método de seleção por torneio foi escrito por Hillol Kargupta, professor da Universidade de Alabama.

Referências Bibliográficas

[ANd96] Delben A., Bretas N., and Carvalho A. de. Restabelecimento Ótimo de energia em sistemas de distribuição: Uma comparação entre o desempenho de busca fuzzy e algoritmos genéticos. Technical Report 48, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Carlos,, Brazil, 1996.

[AW89] K.J. Astrom and B. Wittenmark. *Adaptive Control.* Addison-Wesley, 1989.

[Bel77] N.D. Belnap. A useful four-valued logic. In J.M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logics.* D. Reidel Pub. Co., 1977.

[Bit98a] G. Bittencourt. Concurrent inference through dual transformation. *Logic Journal of the Interest Group in Pure and Applied Logics (IGPL)*, 6(6):795–834, 1998.

[Bit98b] G. Bittencourt. *Inteligéncia Artificial. Ferramentas e Teorias.* Editora da UFSC, Florianópolis, SC, 1998.

[BNW96] P. Bauer, S. Nouak, and R. Winkler. A brief course in fuzzy logic and fuzzy control. Technical report, Fuzzy Logic Laboratorium Linz-Hagenberg, December 1996.

[Boo82] L. B. Booker. Intelligent behavior as an adaptation to the task environment. *Dissertations Abstracts International*, 43(2):469B, 1982. (University Microfilms No. 8214966).

[Bri81] A. Brindle. *Genetic Algorithms for Function Optimization.* PhD thesis, University of Alberta, Edmonton, Canada, 1981.

[Bro86] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):435–453, March 1986.

[Bro91] P.A. Brooks. Intelligence without representation. *Artificial Intelligence (Special Volume Foundations of Artificial Intelligence)*, 47(1-3):139–159, January 1991.

[BS93] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter otimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[CB98] A.C.P.L. da Costa and G. Bittencourt. Dynamic social knowledge: A cooperation strategie for cognitive multi-agent systems. *Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 415–416, Paris, France, July 2-7 1998. IEEE Computer Society.

[Cle94] The Clearinghouse for Genetic Algoritms. *SGA-C: A C-lenguage Implementation of a Simple Genetic Algorithm*, March 1994. AL 35487.

[Coe97] L. dos Santos Coelho. Metodologias da inteligência computacional em identificação e controle de processos: Abordagem nebulosa, evolutiva e neural. Master's thesis, Pós-Graduação em Ciência da Computação- Universidade Federal de Santa Catarina, Fpolis - SC, Fevereiro 1997.

[Cor99] C. Correa. Uso de algoritmos genéticos na construção de controlador nebuloso para o controle de atitude de um satélite artificial durante a fase de apontamento. Master's thesis, Pós-Graduação em Computação Aplicada - INPE, São José dos Campos - SP, Março 1999.

[Dav91] L. Davies. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[Del93] Delft University of Techonology. *RICE, 4.0x*, 1993. MA 02139.

[DP80] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, 1980.

[DP88] D. Dubois and H. Prade. *Possibility Theory - An Approach to the Computerized Processing of Uncertainty.* Academic Press, 1988.

[Fon96] M.Schneider Fontán. Planificación basada en percepción activa para la navegación de un robot móvil. Master's thesis, Instituto de Automática Industrial, Madrid - España, Septiembre 1996.

[FTW83] J.D. Farmer, T. Toffoli, and S. Wolfram. *Cellular Automata: Proceedings of an Interdisciplinary Workshop at Los Alamos.* New Mexico, North-Holland, Amsterdam, March 7-11 1983.

[GH88] D.E. Goldberg and J.H. Holland. Genetic algorithms and machine learning: Introduction to the special issue on genetic algorithms. *Machine Learning*, 3, 1988.

[God97] J. Godjevac. *A method for the design of Neuro-Fuzzy controllers; an application in robot learning.* PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne - Switzerland, 1997.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Reading, MA, 1989.

[Gol94] D.E. Goldberg. Genetic and evolutionary algoritms come to age. *Communications of ACM*, 37(3):113–119, March 1994.

[HB94] J. Heitkoetter and D. Beasley, editors. *The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions.* USENET:comp.ai.genetic. Available via anonymous EMAIL from rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/, 1994.

[JVB94] R. Jager, H. B. Verbruggen, and P. M. Bruijn. Demystification of fuzzy control. In S.G. Tzafestas and A.N.Venetsanopoulos, editors, *Fuzzy Reasoning Information, Decision and Control System*, pages 165–195. Microprocessor-Based Systems Engineering, 1994.

[KJ83] H.E. Kyburg Jr. The reference class. *Philosophy of Science*, 50:374–397, 1983.

[Len95] D.B. Lenat. Cyc, a large-scale investment in knowledge infrastructure. *Communications of ACM*, 38(11):33–44, November 1995.

[MA74] E. H. Mamdani and S. Assiliam. Application on fuzzy algorithm for control of simple dynamic plant. *IEEE Proceedings on Inst. Elec. Eng.*, 121(12):1585–1888, 1974.

[MAG95] MAGE team, CNRS. *Khepera Simulator*, Sep 1995.

[Mat97] Inc. MathWorks. *Fuzzy Logic Toolbox User's Guide.* MathWorks, Inc., 1997.

[McC80] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39, 1980.

[Paw92] Z. Pawlak. Rough sets: a new approach to vagueness. In L.A. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty.* John Wiley & Sons Inc., New York, 1992.

[Pea87] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1987.

[Red95] R. Reddy. Grand challenges in ai. *ACM Computing Surveys*, 27(3):301–303, September 1995.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, April 1980.

[Saf98] A. Saffiotti. *Autonomous Robot Navigation: a fuzzy logic approach.* PhD thesis, IRIDIA - Université Libre de Bruxelles, Bruxelles - France, June 1998.

[San93] S. Sandri. Local propagation of information on directed markov trees. In B. Bouchon-Meunier Et Al., editor, *Uncertainty in Intelligent Systems.* Elsevier Science Publishers, 1993.

[San97] S. Sandri. Introdução à lógica "fuzzy". *II Simposio Brasileiro de Automação Inteligente*, 1997.

[Sch91] R.C. Schank. Where's the ai? *The AI Magazine*, 12:38–49, Dezember 1991. Winter 1991.

[SDP95] S. Sandri, D. Dubois, and H. Prade. Elicitation, pooling and assessement of expert judments using possibility theory. *IEEE Transactions on Fuzzy Systems*, 1995.

[Sha76] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[SM85] M. Sugeno and K. Murakami. An experimental study on fuzzy parking control using a model car. in m. sugeno. In *Industrial Applications of Fuzzy Control*, pages 125–138. North-Holland, 1985.

[TS83] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operator's control actions. In *Proceedings of IFAC Symposium on Fuzzy Information, Knoeledge Representation and Decision Analysis*, pages 55–60. Marseilles-France, July 1983.

[YZ91] R.R. Yager and D.A. Zadeh, editors. *An Introduction to Fuzzy Logic Applications in Intelligent Systems.* Kluwer Academic Publishers, 1991.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad78] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.